Ontology Alignment Mechanisms for Improving Web-based Searching

Ph.D. Thesis

Mohammad Mustafa Taye

This thesis is submitted in partial fulfilment of the

requirement for the Doctor of Philosophy

Awarded by

Faculty of Technology

De Montfort University

United Kingdom, England

2009

Declaration

I declare that the work described in this thesis is original work undertaken by me for the degree of Doctor of Philosophy, at the Software Technology Research Laboratory (STRL), Faculty of Technology, at De Montfort University, United Kingdom. No part of the material described in this thesis has been submitted for the award of any other degree or qualification in this or any other university or college of advanced education.

Dedication

To my beloved parents and family

The thesis is dedicated to my loving father, **Mr. Mustafa Al-Taye**, who has been a great source of motivation, inspiration and endless support throughout my life, and who sacrificed a lot for me to be what I am now.

It is also dedicated to my loving mother, who gave her love and support, for everything she sacrificed in her life for me. Without her loving care, prayers and support, it would have been very difficult for me to achieve my goals.

I owe everything I have achieved or will achieve to them. I hope that by obtaining my PhD I can put smiles on their faces.

ABSTRACT

Ontology has been developed to offer a commonly agreed understanding of a domain that is required for knowledge representation, knowledge exchange and reuse across domains. Therefore, ontology organizes information into taxonomies of terms (i.e., concepts, attributes) and shows the relationships between them. In fact, it is considered to be helpful in reducing conceptual confusion for users who need to share applications of different kinds, so it is widely used to capture and organize knowledge in a given domain.

Although ontologies are considered to provide a solution to data heterogeneity, from another point of view, the available ontologies could themselves introduce heterogeneity problems.

In order to deal with these problems, ontologies must be available for sharing or reusing; therefore, semantic heterogeneity and structural differences need to be resolved among ontologies. This can be done, in some cases, by aligning or matching heterogeneous ontologies. Thus, establishing the relationships between terms in the different ontologies is needed throughout ontology alignment.

Semantic interoperability can be established in ontology reconciliation. The original problem is called the “ontology alignment”. The alignment of ontologies is concerned with the identification of the semantic relationships (subsumption, equivalence, etc.) that hold between the constituent entities (which can be classes, properties, etc.) of two ontologies.

In this thesis, an ontology alignment technique has been developed in order to facilitate communication and build a bridge between ontologies. An efficient mechanism has been developed in order to align entities from ontologies in different description languages (e.g. OWL, RDF) or in the same language. This approach tries to use all the features of ontologies (concept, attributes, relations, structure, etc.) in order to obtain efficiency and high quality results. For this purpose, several matching techniques have been used such as string, structure, heuristic and linguistic matching

techniques with thesaurus support, as well as human intervention in certain cases, to obtain high quality results.

The main aim of the work is to introduce a method for finding semantic correspondences among heterogeneous ontologies, with the intention of supporting interoperability over given domains.

The approach brings together techniques in modelling, string matching, computation linguistics, structure matching and heuristic matching, in order to provide a semi-automatic alignment framework and prototype alignment system to support the procedure of ontology alignment in order to improve semantic interoperability in heterogeneous systems.

This technique integrates some important features in matching in order to achieve high quality results, which will help when searching and exchanging information between ontologies. Moreover, an ontology alignment system illustrates the solving of the key issues related to heterogeneous ontologies, which uses combination-matching strategies to execute the ontology-matching task. Therefore, it can be used to discover the matching between ontologies.

This thesis also describes a prototype implementation of this approach in many real-world case studies extracted from various Web resources. Evaluating our system is done throughout the experiments provided by the Ontology Alignment Evaluation Initiative. The system successfully achieved 93% accuracy for ontology matching. Finally, a comparison between our system and well-known tools is achieved so that our system can be evaluated.

ACKNOWLEDGMENTS

First and foremost, I am thankful to Almighty ALLAH for all his bounties and blessings, for giving me the ability to complete this research. Without him, none of this work would have been possible.

Studying at the University of De Montfort in Leicester and particularly at the STRL was the most rewarding experience I have ever had. To me, this is certainly related to the high standard of the facilities offered to students, the friendly atmosphere among the research students, and above all the excellence of supervision. For this reason I would like first to express my deepest appreciation and lasting gratitude to **Prof. Hussein Zedan**, the head of the STRL. His wide knowledge and his logical way of thinking have been of great value to me. His understanding, encouragement and personal guidance have provided a good basis for the present thesis. Without his guidance the successful completion of the research and this thesis might have been a very difficult task. His critique and helpful ideas showed us the way to proceed. I am really grateful for that. I really appreciate his positive comments for improving and bringing out the optimum consequences from my endeavours. Without his guidance and support I would never have been able to organize and complete my task well and within time constraints. The only thing I can tell him is "thank you for everything".

I would also like to express my thanks and appreciation to my supervisor, **Dr. Antonio Cau,** for his valuable comments, constructive criticism, scientific support, insightful comments, guidance and suggestions, without which this thesis could not have been produced in the present form.

I wish to thank all researchers, colleagues and staff of the **STRL**. During this work I have collaborated with many colleagues for whom I have great regard, and I wish to extend my warmest thanks to all those who have helped me with my work in STRL. I want to thank them for all their help, support and valuable hints. I enjoyed these, as they gave me the feeling of belonging to a group.

I wish to express my love and gratitude to all my family, whose love and support have always been with me. In particular, I would like to give my special thanks to my

beloved parents for their love and their continuing financial and moral support throughout my studies. They had more faith in me than could ever be justified by logical argument. My special gratitude is due to my dearest brothers **Ahmed**, **Mahmoud** & **Ibrahim** and to my lovely sisters and their families for their loving support, concern and encouragement through all these years.

I am exceedingly fortunate in having such warm and generous friends and relatives in Jordan, Palestine, Saudi Arabia, Kuwait, Germany and Italy. Because of the pressures of my studies, I have proved a very bad correspondent, but they did not give up on me. They continually called and emailed to offer their help and support, and thus encouraged me to continue to the end. To them I say thank you for your persistence in keeping in touch with me when I did not reciprocate on so many occasions. Your perseverance is hugely appreciated.

Finally, here at Leicester and especially at De Montfort I have had a great and time and an enjoyable experience. I would like to thank all the good friends whom I have met in Leicester. First, there are two kind people whom I have met in Leicester: **Brigadier Dr Ahmed Al-Ghamdi** and **Dr Osama El-Hassan**. Also special thanks to **Brigadier Ali Alqahtani**, **Nasser Al-Alwan** and **Sameeh Al-Sarayreh.** My special gratitude goes to my loyal friend **Saud Al Otaibi** for his continuing support and always standing at my side. Special thanks are also due to my closest friends and my flatmate, **Dr. Ali Hilal Mohammed**, for his support and contributions. Guys, in this city I have met all of you, so our friendship started here and will continue forever.

Thank you.

Leicester, England, March / 2009

Mohammad Taye

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
DAML	DARPA Agent Markup Language
DL	Description Logic
FOL	First Order Logic
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDF	Inverse Document Frequency
KB	Knowledge Base
KRS	Knowledge Representation System
LSI	Latent Semantics Indexing
NOM	Naive Ontology Mapping
OAEI	Ontology Alignment Evaluation Initiative
OIL	Ontology Inference Layer
OWL	Web Ontology Language
QOM	Quick Ontology Mapping
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
TF	Term Frequency
URI	Uniform Resource Identifier

W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language

Chapter 1 Introduction

This chapter provides an introduction to the thesis, beginning with an outline of the main problem tackled, followed by a statement of the overall goal of the research and its objectives, a description of the way of working and an account of the main contributions made. Finally, the structure of the rest of the thesis is presented.

1.1 Overview

One of the most interesting inventions in recent decades is that of Web services. These computer programs are self-describing, self-contained applications whose function is to share information automatically over the Internet with other applications [89]. However, some weaknesses, such as browsing information without considering its meaning, have recently appeared in Web services. This creates a need for a new Web with more relevance to the user. Indeed, Tim Berners-Lee and his colleagues have described the next generation of the current Web, which they call the Semantic Web [6, 32, 76].

The Semantic Web enables the user and application or computer to work co-operatively, the idea being to enable machines to understand Web pages by adding semantic annotations to them. This means the annotation of the information in a way that helps the machine to understand its meaning, a possibility that could be realised by the addition of an ontology [54, 68, 71].

Accordingly, the Semantic Web is actually an extension of Web services, in that it represents information more meaningfully for humans and computers alike. It enables the description of contents, services and allows annotation, discovery, publishing, advertising and composing services to be automated. Its development was based on ontology, which is considered to be the backbone of the Semantic Web. In other words, the current Web is transformed from machine-readable to machine-understandable.

In fact, ontology is a key technique by which one can annotate semantics and provide a common, comprehensible foundation for resources on the Semantic Web. Ongoing ontology research has become popular in many disciplines, such as computer science, medicine, bioinformatics and geographical information systems. It is considered to be a key component in solving the problem of semantic heterogeneity, thus enabling semantic interoperability between different web applications and services. To this end, several ontologies have recently been built and it should be possible to access them from other applications for use or information exchange.

Ontologies in such numbers present interoperability problems, for which many solutions have been developed. One of these is to build a single ontology, but this is unfeasible, partly because it would be too inflexible for knowledge sharing [6].

Another solution is ontology alignment [74], which plays an important role in ensuring interoperability in heterogeneous systems and in many application domains. Ontology alignment builds bridges between ontologies in order to provide common accessible layers that can then exchange information in semantically sound ways.

Despite the fact that ontologies have became more prevalent in many communities as a means to establish formal and explicit vocabulary that applications can share, it is unrealistic to expect a universal ontology for the World Wide Web (WWW) [89]. Different people have a different vision of the world and consequently people or agents may use different concepts for the similar meaning or may use the same concept to denote different things [6].

In general, different ontology sources use different data formats and modelling languages to represent their data and metadata. After that, sources may use the same data format but differ in their structure and in the semantics of the terminology they use. Such heterogeneity is a result of the independent nature of the ontologies and the fact that different people with different objectives in mind evaluate information sources differently [56].

The problem is that where several ontologies have been developed for the same domain or related fields, the need to compare them, move from one to another or integrate them becomes apparent.

The alignment of ontologies is currently one of the issues at the heart of the popularization of the Semantic Web [89]. An ontology alignment is the expression of relations between different ontologies. Indeed, alignment results can additional support the visualization of correspondence, the transformation of one ontology into another, or the providing of bridge axioms between the ontologies. At present, several systems are available to support users in aligning ontologies, but not a lot of relative estimates have been executed and there exists not much support for performing such evaluations. Several techniques of alignment based on different criteria are proposed in the literature [26, 75]. The choice of one technique or another—or a combination of more than one of them—is not an easy task. Indeed, this choice requires apprehension of the multiple criteria related to current technology alignment and the ontologies considered.

Current Web search engines offer access to thousands of irrelevant results returned as a result of several problems, for example, one word describing several senses or several words describing one sense. Therefore, developing ontologies for web pages t can help by adding more information or semantics to a web page; therefore it can search efficiently by using ontology alignment techniques. Indeed, ontology alignment algorithms were used to provide the correspondences between entities semantically. Also it relies on syntactical techniques in order to anchor the terms to be matched to those found on the Semantic Web. This technique is improved by a necessary information retrieval technique for linguistic, string value and structure matching. Semantic matching requires that the semantics of the query and the document, as well as the content of the Web pages must be analysed, and return the subject of the page in order to discover previous matching between the query and the document. Ontology alignment is significant in reaching semantic interoperability in the WWW. The computation of similarity to discover similarities between ontology terms using combinations of lexical, structural, and semantic knowledge. Using the

thesaurus helped in retrieving relevant information, even if the keyword is not present in a document. All previous steps should be made available into ontology alignment in order to improve the search over the web.

1.2 Motivation and Problem Statement

The vision of the Semantic Web requires a system that can change data and reuse exchanged data with their intended meanings. This is called *semantic interoperability*. Achieving such a state among different information systems is very tedious and difficult. Moreover, errors are inevitable in a distributed and heterogeneous kind of environment such as in different applications and the WWW, which has several billion pages [6].

The heterogeneity of information occurs, in general, at the levels of syntax, structure and semantics. Syntactic heterogeneity is the simplest heterogeneity problem, caused by the use of different data formats. In order to resolve it, standardized formats such as XML [10], RDF/RDFS [11, 51, 60] and OWL [18, 61, 83], have usually been used to express data in a uniform way that makes the automatic processing of shared information easier. At the second level, which is structural, heterogeneity occurs as a result of the way information is structured, even in syntactically homogeneous environments. Indeed, standardization of the format does not in itself overcome such structural heterogeneity. For example, one source may model motor vehicles but classify them into a few categories only, while another may make very fine-grained distinctions among types of vehicle according to criteria such as their physical structure, weight or purpose. Manually encoded transformation rules, as well as some middleware components, have been used to solve structural heterogeneity problems [12, 22, 83].

Finally, the third level, Semantic heterogeneity, occurs for example when two ontologies do not share the same interpretation of information, also for example, when trying to say the same thing in different ways. This level of heterogeneity is still only partially solved, i.e. some approaches have been developed on the basis of

synonym sets [66], term networks [9], concept lattices [86], features and constraints, and have been proposed as solutions for solving semantic heterogeneity among different information systems [26, 47], but they are not sufficient to solve the problem of semantic heterogeneity in the WWW environment.

Ontology alignment involves determining the semantic heterogeneity between two or more domain specifications by considering their associated concepts [46, 71]. The process requires matching, but makes use of more expressive relations between ontology concepts (partOf, subsumes, etc.).

It is generally agreed that such alignment not capable to be done manually beyond a certain complexity, size or number of ontologies. Therefore, in order to decrease the trouble of manual establishment and maintenance of alignments, automatic- or semi-automatic-techniques have to be developed.

Many research studies have recently been done in this area [26, 57, 62, 64, 86], usually on systems which use different techniques to compute or find the similarities between concepts or terms in the different source ontologies. Many of these approaches are discussed later. On the other hand, most existing alignment approaches ignore some crucial realistic aspects of ontology alignment, such as an adequately high quality of results, competence, ability to deal with large size ontology, focus on misconception between ontologies and minimal human effort. Efforts to solve the alignment issue are now being increased with the valid ambition that knowledge integration can be achieved to a much higher degree than has ever been possible before.

1.3 Solution

A framework of multiple strategies ontology alignment have been developed which employs multiple ontology alignment strategies and sets the combination weight manually. Therefore, it relies on a well-established measure for comparing the entities of two ontologies which are combined in a homogeneous way. The Figure 1.1 shows that our prototype comprises three processes:

A. Pre-processing (Feature Generation):

The first step entails obtaining useful information from the ontologies that are to be matched, beginning by loading two ontologies and extracting useful ontological features such as class names and properties. Then normalization is carried out on these elements, by removing stop words, for example.

B. Alignment Process (Group of Matchers):

In general, the similarity between entities needs to be calculated in order to find the correspondence between ontology entities. For that reason, different strategies were used (e.g. string similarity, synonyms, structural similarity and similarity based on instances) for achieving similarity between entities.

C. Post -Processing:

- Similarity Aggregator.
- Similarity Evaluator.

The system starts by loading two ontologies and extracts useful features such as class names, property names and subsumption relationships from them. The second step is applying different matching algorithms. The following is classifying of the matching-strategies that are defined in our system:

- **String Matching:** This method is used to compute similarities based on the names of class and property. It can also be used to compute the similarity between two classes by computing the similarities between the names of their properties.
- **Linguistic-based Strategies** match the strings with a thesaurus in order to obtain synonyms and hyponyms.

- **Structural Matching** is a method used to compute the similarity between two classes by using graph information like compute the similarity between the super-classes of the two classes.

- **Heuristic-based Strategies:** In this kind of matching, several features of the string matcher were combined with those of iteration, computing the similarity in order to achieve high quality results.

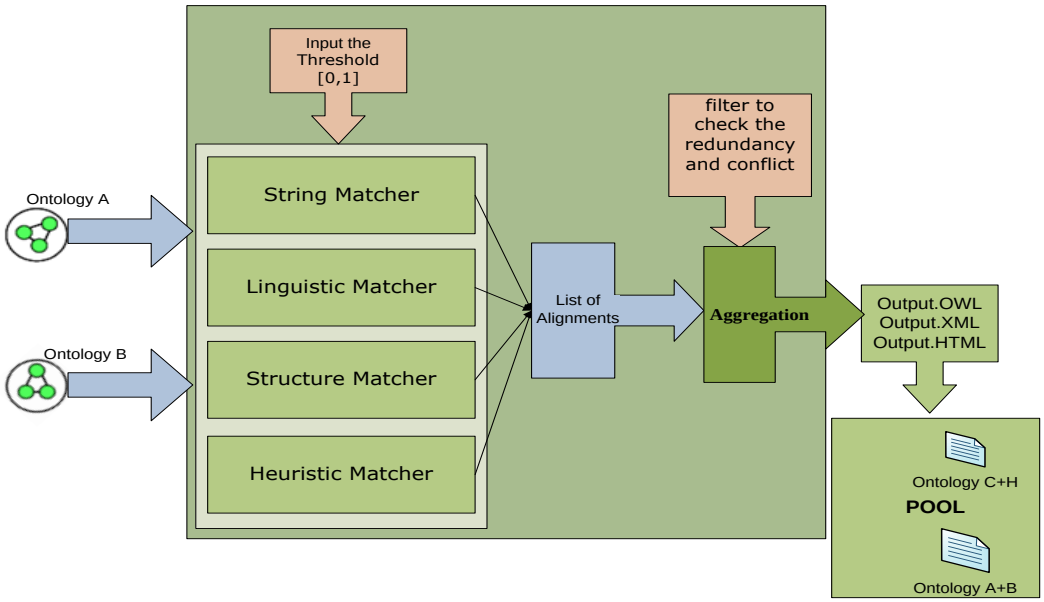


Figure 1.1: General Alignment Framework.

Finally, the outcome from all matching steps is a set of alignment entities, which will be aggregated by efficient algorithms to check the correctness of alignment entity relationships and avoid redundancy.

1.4 Methodology

What kind of methodology should be used to evaluate the correctness and completeness of the alignment?

In fact, there is no simple measure available to evaluate the correctness and completeness of the alignment, because either the size of the ontologies is too large to find a complete set of matching manually, or it is too difficult to define matching precisely. Therefore, most ontology engineers adopt the standard precision, recall and f-measure methods from information retrieval research to evaluate matching or alignment results [28]. On the other hand, a few researchers evaluate matching results by counting the number of adjustments necessary between the matches found and the reference ontology [65]. In the work reported in this thesis, we adopted the first approach, taking precision, recall and f-measure as our evaluation criteria, consistent with the approach used by other researchers. Moreover, the alignment results were represented as a list of matching pairs, following the format required by the annual Ontology Alignment Evaluation Initiative (OAEI) ontology matching campaign [96].

1.5 Research Questions

Recently, theoretical and practical work [26, 75] has been reported on proposed methods of alignment. On the other hand, the available alignment systems do not satisfy the following requirements, on which our system will focus:

1. **What are the factors that most strongly influence the number of ontologies that can be aligned**?

The answer to this question leads to the development of a framework for ontology alignment which stresses the importance of hierarchies in the representation of models of ontology. The framework also makes use of most ontology features. In fact, ontologies capture the concepts, structure, relationships, semantics and other necessary meta-information of an application. Therefore, all the information that is useful to derive matching should be studied by the approach. Consequently, there is a

need for making use of instance information (if available) to enhance the matching process, because instance level can provide important regarding the contents and meaning of ontology elements.

2. **How can alignment achieve high quality results**?

The answer to this question is multiple matching techniques are combined to obtain high quality results; these are string-, structure-, linguistic- and heuristic-based techniques.

3. **How can alignments be achieved competently and efficiently**?

Very efficient calculation of alignments can be achieved by using most features of ontology to select the most promising alignment candidates for comparison. The efficiency in our system is achieved by choosing perfect and tractable matching techniques. This allows us to reuse one ontology alignment approach for many scenarios.

4. **Why is the flexibility in alignment methods necessary**?

Alignment methods need to be flexible enough to be transferred to other applications, domains and even types of structure. Therefore, multiple matching techniques are combined in order to deal with different domains.

In term of user interaction: allows user intervention to choose the ontologies to be aligned and lets the user accept, delete and add matching entities to the final set of alignments.

5. **How can different matching methods be adapted to deal with different sets of features or to cope with ontology mismatching**?

This is done through an efficiency aggregation algorithm.

1.6 Contributions of the Study

The contributions of the study reported in this thesis can be briefly summarised as follows:

The main contribution of this thesis is the development and specification of an approach to ontology alignment. This effort has been directed in order to improve interoperability across heterogeneous systems, by:

A. Original Contributions are:

- Methodology for designing and developing an ontology alignment approach to discover the semantic correspondences between terms in different ontologies. Also, a method for semantic enrichment and discovery of semantic correspondences between ontologies has been proposed, which contributed to the understanding of semantic distance between ontologies in general.

- Improving a semantic matcher based on combining lexical matcher with several rules and facts.

- Developing a new heuristic matcher which provided very high quality results.

- Developing techniques that were shown to work over several domains and to provide correct results by using most features of ontologies.

- Multiple matching techniques were combined in order to obtain high quality results; these techniques were string-, structure-, linguistic- and heuristic-based. In fact, different matching methods have been adapted to deal with different sets of features and to cope with ontology mismatching.

- Implementing all components of our system. Also, illustrating examples of each component. To this end, a very efficient calculation of alignments can be achieved by using most features of ontology to select the most promising alignment candidates for comparison. This allows the reuse of one ontology alignment approach for many scenarios.

- Providing an analysis of the implementation and evaluation of the method in empirical experiments, presenting the results of the validation experiments that evaluated our system against top ranking competitors.

- Achieving high quality results with efficiency throughout applying the system to real-world scenarios. Also, achieving high scores in terms of both accuracy and completeness when applying the system to ontologies rich with linguistic information.

- Illustrating the application of our system to a case study (a multi-agent system) and showing its ability to deal with this case.

B. The others contributions are:

- An in–depth review to the definition and structure of ontology. Also, various operations over ontologies and a different set of ontology matching methods have been proposed

- A detailed review state-of-the-art of the ontology languages which are used to express ontology over the Web; all these terms were shown in order to provide a basic understanding of ontologies and of description logics, which are the basis of ontology languages.

- A comprehensive review of existing ontology alignment tools. Several existing ontology mapping methods were analysed and compared, since before creating a new approach it is essential to fully understand related work. Here, this means both theoretical work and existing approaches to the alignment of ontologies or other well-defined structures, including their weaknesses, which must be understood if they are to be improved.

- Exploring multiple similarities, such as string-based similarity, profile similarity and structural similarity, to support ontology mapping.

1.7 Criteria for Success

The following criteria are disposed to assess the success of the research delineated in this thesis.

- What are the main elements and important factors that most strongly influence the number of ontologies that can be aligned?
- How would one evaluate tools that can work effectively over several domains?
- Is it important to achieve high quality results from ontology alignment or just normal results?
- In what terms is it important to achieve ontology alignments competently and efficiently? Therefore, in what applications has the system been used successfully? Where has it failed?
- What does flexibility mean in the ontology alignment methods and how could it be achieved? Also, why it is needed?
- Based on what could the value of threshold be chosen?
- How could the results of different matching methods be aggregated to provide results that express the right result?
- What is the importance of assigning weight to each matcher when the tool used several strategies to provide a matching between ontologies?

1.8 Thesis Outline

The remainder of this thesis is structured as follows.

Chapter 2, Overview of Ontology, provides an overview of Web services and Web semantics, and sets out the definition, structure and some applications of ontology

also provides definitions of ontology mapping and other operations, such as ontology alignment and ontology merging. It then offers a clear description and comparison of ontology description languages such as RDF(S), OIL+DAML and OWL. It also describes Description Logic (DL) and some ontology editors.

Chapter 3, Ontology Matching Techniques and Mapping Algorithms, discusses mismatching between ontologies and presents existing matching techniques, discussing which of them will help in obtaining a high quality result and why. Finally, it considers several tools or approaches used to map between ontologies.

Chapter 4, Ontology Alignment System, outlines the most important features and elements required by an alignment system. It also provides a full description of the main components of our proposal and explains how they will interact to provide good mapping results.

Chapter 5, Implementation of the System, introduces a full implementation of our system with some small case studies to provide a full picture of the present approach and show its ability to produce high quality results.

Chapter 6, Evaluation of Results, shows that the proposed system is sufficiently mature to deal with different real-world scenarios; it provides a comparison between our system with top-ranked systems on the Benchmark Test in the OAEI Campaign 2007.

Chapter 7, Conclusion and Future Work, summarises the results, draws conclusions and offers suggestions for future work.

Chapter 2

Semantic Web and Ontologies – State Of the Art

2.1 Introduction

This chapter introduces the basic concepts of web services and the Semantic Web, defines the structure and the main applications of ontology, and provides a brief survey of ontology languages which are used to express ontology over the web. All the relevant terms are explained to provide a basic understanding of ontologies, which are the basis of this work.

One function of the Web is to build a source of reference for information on several subjects, while the Semantic Web is designed to build a web of meaning. The foundation of vocabularies and effective communication on the Semantic Web is ontology. "*Ontology provides a formal, explicit specification of a shared conceptualisation of a domain*" [76, 89]. Therefore, it facilitates knowledge sharing over distributed systems; in other words, it allows systems or applications to cooperate that were not formerly designed to interoperate.

2.2 Web Service

A web service [89] is a self-describing software program using self-contained applications and identified by a Uniform Resource Identifier (URI), used to share information between applications over the Internet. Access to and retrieval of information from the Web occurs via the HTTP protocol. One of the first languages to have been used for the internet is HTML, a markup language used to describe the document structure. The Web can be conceived as a huge library containing a large amount of information or data – unfortunately without any sensible means of representation.

The common Web service scenario [89] can be described by the three actions of publish, bind and find, and three actors: the service requester, the service provider and the registry, where services can be published, advertised and sometimes located. In other words, service providers describe and advertise their services in the registry, while service requesters search the registry for services that match their requirements.

2.3 Semantic Web

The Semantic Web [89] is distributed and heterogeneous, has brought the evolution of the Web to a higher level. There are two visions of the future in the development of the Web, the first being to improve its usability as a medium for collaboration and the second to ensure that its contents can be understood by machines. Providing annotation data will facilitate this second aim.

Tim Berners-Lee, who invented the WWW and has worked on the Semantic Web, states that the latter "is *an extension of the current Web, in which information is given a well-defined meaning, better enabling computers and people to work in cooperation*." [6]. Thus, the Semantic Web [32, 76] is distinguished by a more meaningful representation of information for humans and computers, providing a description of its contents and services in machine-readable form; moreover, it enables services to be automatically annotated, discovered, published, advertised and composed. It thereby facilitates interoperability and the sharing of knowledge over the Web. Its main goal is therefore to make information on the Web accessible and understandable by humans and computers. Figure 2.1 illustrates the architecture of the Semantic Web.

In fact, both the Semantic Web and Web services are considered to be a set of resources, identified by the URI. The difference between them is that Web services use HTTP to display the contents of a page, while the Semantic Web tries to create machine readability by semantically representing the data or information in resources. Numerous tools and applications of Semantic Web technologies have recently become available.

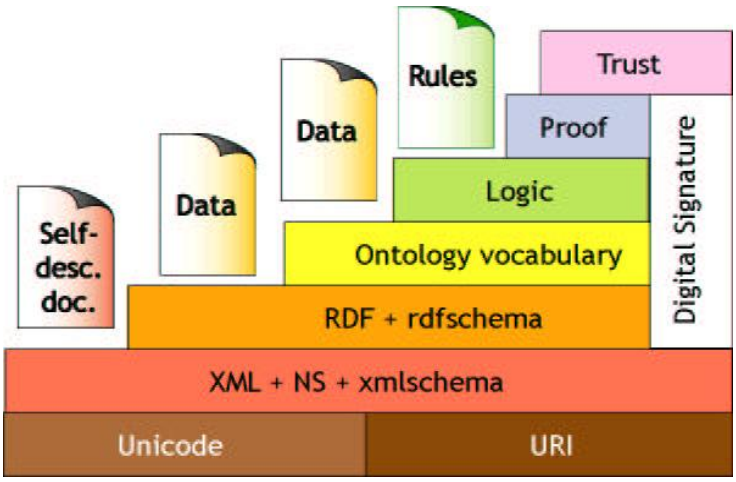


Figure 2.1: Semantic Web Architecture [6]

The layers of architecture represented [6, 32, 76, 89] in Figure 2.1 are briefly described below:

I. **URI and Unicode:** To identify and locate resources, or indeed anything on the Web, a uniform system of identifiers (URIs) is used. The URI, which is considered to be the foundation of the Web, is used to give a unique name to each resource. Unicode is the standard for computer character representation.

II. **Extensible Markup Language** (XML) is a markup language, which means that it is machine-readable and has its own format. It is widely known in the WWW community because it has a flexible text format and was designed to describe data and to meet the challenges of large-scale e-business and electronic publishing; it plays an important role in exchanging different types of data on the Web. In fact, it is the basis of a rapidly growing number of software development activities. Each document starts with a namespace declaration using XML Namespace.

III. **The Resource Description Framework** (RDF) is the first layer of the Semantic Web. RDF is a framework for using and representing metadata and describing the semantics of information about resources on the Web in a

machine-accessible way. It uses URIs to identify Web resources and to describe the relations between these resources, using a graph model. While describing classes of resources and the properties between them, using RDF Schema (which is a simple modelling language), it also provides a simple reasoning framework for inferring types of resources.

IV. **Ontology Vocabulary** is a language which provides a common vocabulary and grammar for published data as well as a semantic description of the data used to preserve the ontologies and to keep them ready for inference. Ontology means describing the semantics of the data, providing a uniform way to enable communication by which different parties can understand each other.

V. **Logic and Proof:** In the Semantic Web, the building of systems follows a logic which considers the structure of ontology. A reasoner could be used to check and resolve consistency problems and the redundancy of the concept translation. A reasoning system is used to make new inferences.

VI. **Trust** is the final layer of the Semantic Web. This component concerns the trustworthiness of the information on the Web in order to provide an assurance of its quality.

2.4 Ontology

Ontologies, which are used in order to support interoperability and common understanding between the different parties, are a key component in solving the problem of semantic heterogeneity, thus enabling semantic interoperability between different web applications and services.

Recently, ontologies have become a popular research topic in many areas, including electronic commerce, knowledge management, knowledge engineering and natural language processing. Ontologies provide a common understanding of a domain that can be communicated between people, and of heterogeneous and widely spread

application systems. In fact, they have been developed in Artificial Intelligence (AI) research communities to facilitate knowledge sharing and reuse.

The goal of an ontology is to achieve a common and shared knowledge that can be transmitted between people and between application systems. Thus, ontologies [32] play an important role in achieving interoperability across organizations and on the Semantic Web [89], because they aim to capture domain knowledge and their role is to create semantics explicitly in a generic way, providing the basis for agreement within a domain. Ontology is used to enable interoperation between Web applications from different areas or from different views on one area. For that reason, it is necessary to establish mappings among concepts of different ontologies to capture the semantic correspondence between them. However, establishing such a correspondence is not an easy task [76].

Because there are many different definitions of ontology, the present research first presents some of these definitions which have been given from different perspectives, and then explores in depth those aspects of these definitions which are related to the topic under investigation.

The primary use of the word "ontology" is in the discipline of philosophy, where it means "*the study or theory of the explanation of being*"; it thus defines an entity or being and its relationship with and activity in its environment. In other disciplines, such as software engineering and AI, it is defined as "*a formal explicit specification of a shared conceptualization*" [68]. The foundations of this definition are:

- All knowledge (e.g. the type of concepts used and the constraints on their use) in ontology must have an explicit specification.
- An ontology is a conceptualisation, which means it has a universally comprehensible concept.
- "Shared" indicates an agreement about the meaning in such domains. In other words, an ontology should capture consensual knowledge accepted by the communities.

- "Formal" refers to the grounding of representation in well understood logic, and any ontology should be machine-processable.

2.4.1 Ontology Representation

Ontology is comprised of four main components: concepts, instances, relations and axioms. The present research adopts the following definitions of these ontological components:

- **A Concept** (also known as a class or a term) is an abstract group, set or collection of objects. It is the fundamental element of the domain and usually represents a group or class whose members share common properties. This component is represented in hierarchical graphs, such that it looks similar to object-oriented systems. The concept is represented by a "super-class", representing the higher class or so-called "parent class", and a "subclass" which represents the subordinate or so-called "child class". For instance, a university could be represented as a class with many subclasses, such as faculties, libraries and employees.

- **An Instance** (also known as an individual) is the "ground-level" component of an ontology which represents a specific object or element of a concept or class. For example, "Jordan" could be an instance of the class "Arab countries" or simply "countries".

- **A Relation** (also known as a slot) is used to express relationships between two concepts in a given domain. More specifically, it describes the relationship between the first concept, represented in the domain, and the second, represented in the range. For example, "study" could be represented as a relationship between the concept "person" (which is a concept in the domain) and "university" or "college" (which is a concept in the range).

- **An Axiom** is used to impose constraints on the values of classes or instances, so axioms are generally expressed using logic-based languages such as first-order logic; they are used to verify the consistency of the ontology.

2.4.2 Structure of Ontology

In general, the structure of an ontology [71] is described as a

$$5\text{-}tuple\ O\text{:} = (C, H^C, R, H^R, I),$$

Where

- C represents a set of concepts (instances of "*rdf:Class*"). These concepts are arranged with a corresponding subsumption hierarchy H^C.
- R represents a set of relations that relate concepts to one another (instances of "*rdf:Property*"). $R_i \in R$ and $R_i \rightarrow C \times C$.
- H^C represents a concept hierarchy in the form of a relation (a binary relation corresponding to "*rdfs:subClassOf*"). $H^C \subseteq C \times C$, where $H^C (C_1, C_2)$ denotes that C_1 is a subconcept of C_2.
- H^R represents a relation hierarchy in the form of a relation $H^R \subseteq R \times R$, where $H^R (R_1, R2)$ denotes that R_1 is a subrelation of R_2("*rdfs:subPropertyOf*").
- I is the instantiation of the concepts in a particular domain ("*rdf:type*").

In general, there are many ways to represent or model the classification of concepts semantically. These include taxonomies, thesauri and ontologies. These widely varying concepts are used in Web semantics, which is why it is necessary to apply taxonomy, or the science of identifying and arranging vocabulary in the shape of a hierarchy or tree. In other words, it is used to describe concepts and their relationships explicitly. The relationships of "subclass" and "super-class" are the taxonomic ones that could be used.

A thesaurus [66] is a vocabulary with conceptual relationships between the terms and can be considered an extension of taxonomy with richer semantic relationships. It can easily be converted into a taxonomy, but expressiveness and semantics will be lost.

The relationships which could be used in a thesaurus are equivalence, homography, hierarchy and association.

Ontologies are like taxonomies but with more semantic relationships between concepts and attributes; they also contain strict rules used to represent concepts and relationships. An ontology is a hierarchically structured group of concepts for describing a domain that can be used as a skeletal foundation for a knowledge base.

The ontology community distinguishes ontologies into *lightweight* and *heavyweight* ontologies. The former include concepts, concept taxonomies, relationships between concepts, and properties that describe concepts, while heavyweight ontologies add axioms and constraints to lightweight ones. These clarify the intended meaning of the terms gathered in the ontology.

Heavyweight and lightweight ontologies can be implemented in various kinds of languages [43]. Ontologies can be:

- *Highly informal* if they are expressed in natural language; According to this, a highly informal ontology would not be an ontology, since it is not machine-readable.

- *semi-informal* if expressed in a restricted and structured form of natural language, since it is a machine-readable;

- *semi-formal* if expressed in an artificial and formally defined language (e.g. RDF graphs); and

- *Rigorously formal* if they provide meticulously defined terms with formal semantics, theorems and proof of properties such as soundness and completeness (e.g. Web Ontology Language [OWL]).

The expressiveness of an ontology is based on the degree of explication of the (meta-) knowledge. Several ontologies capture specific domains or certain applications, while others try to capture all terms in natural language. Ontologies that capture extra relations and extra constraints are considered to be more expressive, because they

capture knowledge of the domain on a more detailed level. On the other hand, the expressiveness of an ontology is restricted by the languages used for describing or specifying it. Ontology languages can be seen as restricting the expressiveness of the ontology [91].

An ontology is expressed in a knowledge representation language, which provides a formal frame of semantics. This ensures that the specification of domain knowledge in an ontology is machine-processable and is being interpreted in a well-defined way [6].

2.4.3 Ontology Applications

Over the years, ontology has become a popular research topic in a range of disciplines, with the aim of increasing understanding of and building a consensus in a given area of knowledge. Ontology also leads to the sharing of knowledge between systems and people. As mentioned previously, ontology first appeared in AI laboratories, before being used in other fields; for example:

I. **Semantic Web** [6]: Ontology plays a key role in the Semantic Web in supporting information exchange across distributed environments. The Semantic Web represents data in a machine-processable way, which is why it is considered to be an extension of the current Web.

II. **Semantic Web Service Discovery** [6]: In the e-business environment, ontology plays an important role by finding the best match for the requester looking for merchandise or something else. It also helps online travel customers obtain a response.

III. **Artificial Intelligence** [68]: Ontology has been developed in the AI research community, its goal here being to facilitate the sharing of knowledge and the reuse and enabling of processing between programs, services, agents or organisations across a given domain.

IV. **Multi-agent** [32]: The importance of ontology in this area is that it provides a shared understanding of domain knowledge, allowing for easy communication between agents and thereby reducing misunderstandings.

V. **Search Engines** [32, 76]: These use ontology in the form of thesauri to find the synonyms of search terms, which facilitates internet searching.

VI. **E-Commerce** [32]: This application uses ontology to facilitate communication between seller and buyer through the description of merchandise, as well as enabling machine-based communication.

VII. **Interoperability** [16]: The problem of bringing together heterogeneous and distributed systems is known as the "interoperability problem". In this area, the importance of applying ontology appears explicitly: it is used to integrate different heterogeneous application systems.

In the field of services, ontology plays the major role of providing a richer description of these services and terms and the relationships between them in the application domain, leading to a capture of the domain of knowledge in an explicitly representative manner. At the same time, it supports the inference of implied knowledge by declaring the descriptions.

The following example is given in order to demonstrate the reasons for considering ontology to be the backbone of the Semantic Web. As mentioned in [54], it illustrates how ontology may be used to match services with semantic meanings. According to this scenario, the service requester invokes a service (for example, buying a car), which triggers a description of the service request information annotated in metadata. Service providers also describe and advertise their services in metadata to provide answers to the requester, while the service match engine receives the metadata of both provider and requester, upon which it accesses the ontology, which provides a possible identification of "automobile" and "vehicle" with "car". The service match engine will infer from this whether the request has been satisfied or not (see Figure 2.2).

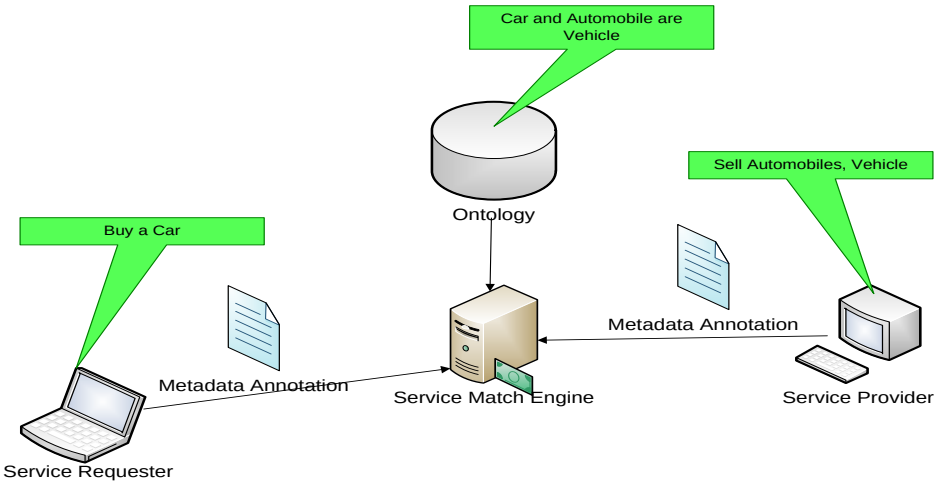


Figure 2.2: Ontology in Service over the Internet

2.4.4 Ontology Interoperability

This section describes several operations on ontologies.

2.4.4.1 Ontology Transformation and Translation

Ontology Transformation [14, 22] is the process used to develop a new ontology to cope with new requirements made by an existing one for a new purpose, by using a transformation function t. In this operation, many changes are possible, including changes in the semantics of the ontology and changes in the representation formalism.

Ontology Translation is the function of translating the representation formalism of an ontology while keeping the same semantic. In other words, it is the process of change or modification of the structure of an ontology in order to make it suitable for purposes other than the original one.

There are two types of translation. The first is translation from one formal language to another, for example from RDFS to OWL, called syntactic translation. The second is translation of vocabularies, called semantic translation [14].

The translation problem arises when two Web-based agents attempt to exchange information, describing it using different ontologies.

2.4.4.2 Ontology Merging

Ontology merging [41, 48, 75] is the process of creating a new single coherent ontology from two or more existing source ontologies related to the same domain. The new ontology will replace the source ontologies.

2.4.4.3 Ontology Integration

Integration [48, 75] is the process of creating a new ontology from two or more source ontologies from different domains.

2.4.4.4 Ontology Mapping

Ontology mapping [40, 47, 73, 78, 82] is a formal expression or process that defines the semantic relationships between entities from different ontologies. In other words, it is an important operator in many ontology application domains, such as the Semantic Web and e-commerce, which are used to describe how to connect and from correspondences between entities across different ontologies.

An entity *e* is understood in an ontology *O* denoted by $e|O$ is concept C, relation R, or instance I, i.e. $e|O \in C \cup R \cup I$. Mapping the two ontologies, O1 onto O2, means that each entity in ontology O1 is trying to find a matching entity which has the same intended meaning in ontology O2 [40].

The Ontology mapping function "map" is determined based on set of E, of all terms $e \in E$ and based on the set of possible ontologies, O as a partial function:

$$\text{map: } E \times O \times O \rightharpoonup E, \textit{ with}$$

$$\forall e \in O1(\exists\ f \in O2 : \text{map}(e,O1,O2) = f \ \vee \text{map}(e,O1,O2) = \perp).$$

An entity is mapped to another entity or none.

2.4.4.5 Ontology Alignment

Ontology alignment [27, 29, 52] is the process or method of creating a consistent and coherent link between two or more ontologies by bringing them into mutual agreement. This method is near to artificial intelligence methods: being a logical relation, ontology alignments are used to clearly describe how the concepts in the different ontologies are logically related. This means that extra axioms illustrate the relationship between the concepts in different ontologies without changing the meaning in the original ontologies.

In fact, ontology alignment uses as pre-process for both ontology merging and ontology integration.

There are many different definitions of ontology alignment, depending upon its application and its intended outcome. Sample definitions include the following:

1. Ontology alignment is used to "*establish correspondences among the source ontologies, and to determine the set of overlapping concepts, concepts that are similar in meaning but have different names or structure, and concepts that are unique to each of the sources"* [75].

2. *"Ontology alignment is the process of bringing two or more ontologies into mutual agreement, making them consistent and coherent"* [16, 25, 81].

3. "*Given two ontologies O1 and O2, mapping one ontology onto another means that each entity (concept C, relation R, or instance I) in ontology O1 is trying to find a corresponding entity (i.e. by using matching algorithms), which has the same intended meaning, in ontology O2*" [26].

Formally, an ontology alignment function is defined as follows:

An ontology alignment function, align, based on the set E of all entities $e \in E$ and based on the set of possible ontologies O, is a partial function.

$$\text{Align: } O1 \rightarrow O2$$

$$\text{Align } (e_{O1}) = f_{O2} \text{ if } \text{Sim}(e_{O1}, f_{O2}) > \text{threshold.}$$

Where Oi: ontology, $e_{Oi,}$ f_{Oj}: entities of (Oi, Oj) Sim (e_{O1}, f_{O2}): similarity function between two entities e_{O1} and f_{O2}.

The ontology alignment function is based on different similarity measures.

A similarity measure is a real-valued function Sim (e_i, f_j): O×O → [0, 1] measuring the degree of similarity between x and y.

$$Sim(e_i, e_j) = \begin{cases} 1 & e_i = e_j \quad two\ entities\ are\ identical \\ 0 & e_i \neq e_j \quad two\ entities\ are\ different \end{cases}$$

2.4.4.6 Other Ontology Operations

There are many operations that could apply to ontology, such as changing [48], which is considered one of the most interesting and important operations that should be taken into account when dealing with ontology. In general, most existing ontologies have large sizes and complex structures. In fact, several factors could be responsible for a change in ontology, including a response to new needs or requirements, a change by the developer or the editor of ontology, an ontological translation from one language to another and a change of domain of interest. On the other hand, using versioning could help to reduce those problems by keeping track of the relationships between different revisions of ontology [48].

2.5 Web-Based Ontology Languages

2.5.1 Introduction

The main object of semantic web languages is to add semantics to the existing information on the Web. RDF/RDFS [11], OIL [31], DAML+OIL [34] and OWL [1] are modelling web languages that have been developed to represent or express ontologies. In general, most of them are based on XML [10] syntax, but they have different terminologies and expressions. Indeed, some of these languages have the

ability to represent certain logical relations which others do not. Because some languages have greater expressive power than others, their selection for representing ontologies is based mainly on what the ontology represents or what it will be used for. In other words, different kinds of ontological knowledge-based applications need different language facilitators to enable reasoning on ontology data. These description languages provide richer constructors for forming complex class expressions and axioms.

In fact, most recent ontology developers have used ontology editors, which are environments or tools used directly for editing, developing or modifying ontologies. They are used to provide support to the ontological development process, as well as to conceptualise the ontology; they transform the conceptualisation into an executable code using translators. Therefore, the output ontology of these tools will be in one of the Web ontology languages supported by editors such as Protégé [75], OWL-P [20] and OilEd [9]. Alternatively, ontology reasoners are used to check the conflicts with a high degree of automation. Many such systems have recently been developed, including RACER [33] and FaCT [90].

Returning to the main concern of this section, modelling web languages, there are in general two different types: presentation languages such as HTML, designed to represent text and images to users or requesters without reference to the content, and data languages, intended to be processed by machines. The present research relates to the latter.

Before OWL, much research had been conducted into creating a powerful ontology modelling language. This research stream began with the XML-based RDF and RDF/S, progressed to the Ontology Inference Layer (OIL) and continued with the creation of DAML+OIL, the result of joining the American proposal DAML-ONT5 with the European language OIL. All these languages are based on XML or RDF syntax and are consequently compatible with web standards. Indeed, RDF and OWL make searching for and reusing information both easier and more reliable, because they are considered as standards that enable the Web to be a global infrastructure for sharing documents and data equally.

As mentioned in [1], some important requirements for quality support should be taken into account when developing languages for encoding ontologies. These include giving the user explicit written format, ease of use, expressive power, compatibility, sharing and versioning, internationalisation, formal conceptualisations of domain models, well-defined syntax and semantics, efficient reasoning support, sufficient expressive power and convenience of expression.

Syntax is one of the most important features of any language, so it should be well-defined; it is also the most significant condition required for the processing of information by machine.

The semantics of knowledge should be well defined, because it represents the meaning of that knowledge. Formal semantics should be established in the domain of mathematical logic in a clearly defined way that will lead to unambiguous meaning, since well defined semantics will lead to correct reasoning. Semantics can be considered a prerequisite to support reasoning. On the other hand, reasoning will help to check and discover consistent ontology, to verify unintended relationships between classes and to classify individuals into classes.

This section has detailed the most common and important languages, RDF, RDF/S, DAML+OIL and OWL, all of which are based on XML. XML itself [10] is widely known in the WWW community, because it is a flexible text format designed to describe data and to meet the challenges of large-scale e-business and electronic publishing, which plays an important role in exchanging different types of data on the Web. In fact, it is the basis of a rapidly growing number of software development activities.

2.5.2 Ontology Description Languages

Ontology language is the basis of ontological knowledge systems, the definition of a system of knowledge representation language specification; it not only has a rich and intuitive ability to express and use it, but the body should be easily understood by the computer, processing and applications.

On 10 February, 2004, the World Wide Web Consortium (W3C) announced its support for two Semantic Web technology standards, RDF and OWL; that is, the information resources described in semantic language specification. OWL is a standard ontology description language, built on the RDF, which is based on the XML-authoring tools, used mainly to express the needs of computer applications to deal with knowledge and information in the document.

2.5.2.1 Resource Description Framework (RDF)

RDF [4, 51, 60], a language used to provide a standard for metadata about the resources on the Web, is capable of representing data on and exchanging knowledge over the Web. It was developed to be understood by computers, facilitating interoperability between applications. In other words, it is a framework for using and representing metadata and describing the semantics of information about web resources in a way that is accessible to machines.

RDF, which is recommended by the W3C, uses URIs to identify resources or things (the root of an ontology is called a thing). It is based upon XML, which is designed for syntax, while RDF is intended for semantics. As has been mentioned, it is a framework for describing web resources, which is why it has become a common method of describing the properties, time, information and content of web resources, so that it can be read and understood by computer applications.

RDF can be used in several applications, one of the most important being resource discovery, used to enhance search engine capabilities. It is also used to facilitate knowledge sharing and exchange in intelligent software agents and, as previously mentioned, to describe the content and content relationships available with any resource, such as a page.

The RDF model has three elements: a resource (the subject), the object and the predicate. It is possible to say that <subject> has a property <predicate> valued by <object>.

For example, a triplet could be "H. Zedan is the Head of the STRL Group". In an RDF graph, all triplets "nodes and arcs" should be labelled with qualified URIs. In this example, it could be said that the subject (resource) is the STRL Group, the predicate (property) is Head of, and the object (literally) is H. Zedan.

2.5.2.2 Resource Description Framework Schema (RDFS)

The Resource Description Framework Schema (RDFS) [11] has been built upon the XML and RDF models and upon syntax. RDFS offers extra facilities to encourage evolution in both the individual RDF vocabularies and the core RDF Schema vocabulary.

It provides a machine-understandable system for defining the vocabularies needed for such applications or descriptive vocabularies. In other words, it is a group of RDF resources that can be used to define or express the properties of other RDF resources which define application-specific RDF vocabularies. At the same time, RDF(S) helps developers to describe classes and properties in a specific way and to specify relationships between those properties and classes, allowing combinations between classes, properties or values. In other words, RDFS is used to define RDF vocabularies.

In general, RDFS is defined in a namespace informally called 'rdfs' and identified by the URI reference http://www.w3.org/2000/01/rdf-schema#. On the other hand, RDF is defined in a namespace informally called ‘rdf’ and identified by the URI reference http://www.w3.org/1999/02/22-rdf-syntax-ns#.

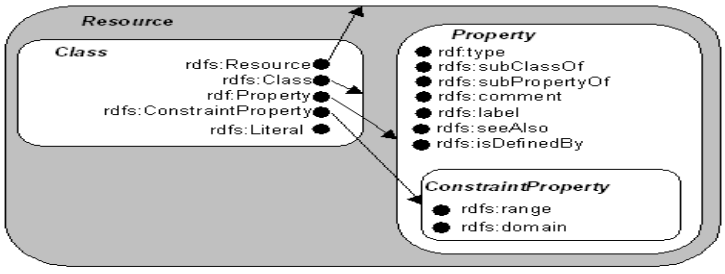


Figure 2.3: Elements of RDF and RDFS [22]

Figure 2.3 presents a description of RDF /RDFS elements, demonstrating three important cores: class, property and constraint property.

In general, an RDF document contains two lists, one of descriptions and one of properties, each of which relates to one resource. Property values could be URIs, literals or other descriptions.

The rdf:RDF includes a sequence of XML elements called rdf:Description; there are “rdf:about” and “rdf:ID”. In any source should use only one of those attributes. rdf:about is used to describe any resource; its value, whether an absolute or a relative URI. rdf:ID, is used to define a resource, so its value of a fragment to be added to the XML document URI.

The elements of RDF are Resource, Property and Property Value: first, Resource is anything that can have a URI, http://www.dmu.ac.uk/RDF; Property is for named resources such as "university"; and Property Value is the value of a property, such as "De Montfort". For example:

```
<?xml version="1.0"?>
 <RDF>
  <Description about="http://www.dmu.ac.uk/RDF">
    <university>De Montfort</university>
    <location>UK-Leicester</location>
  </Description>
</RDF>
```

RDF [51] is a data format that can be presented as an RDF graph. RDF data are proposed in triples of the following components:

- Subject: the entity with which the data deals.
- Predicate: a characteristic of the subject at issue.
- Object: the value given to the predicate.

The statement of RDF is a combination of a resource, a property and property value forms, as in the statement: "The University of Http://www.dmu.ac.uk/RDF is De Montfort".

- The subject of the statement above is: http://www.dmu.ac.uk/RDF.
- The predicate is: university.
- The object is: De Montfort.

Statement: "The location of http://www.dmu.ac.uk/RDF is UK-Leicester".

- The subject of the statement above is: http://www.dmu.ac.uk/RDF.
- The predicate is: location.
- The object is: UK-Leicester.

Although RDF is a good basic language for building many other languages, it is not very expressive and has limitations in describing resources, including descriptions of existence, cardinality, localised range and domain constraints or transitive, inverse or symmetrical properties. In general, as mentioned in [1], the expressive power of RDFS is limited; on other hand, RDF/ RDFS provide modelling that concerns the organisation of vocabularies in terms of hierarchies: subclass and sub-property relationships, domain and range restrictions, and instances of classes. However, some features, such as the specialised or defined properties of local scope and the specialisation of their characteristics of properties, are still missing. It is impossible to separate some classes from each other. For example, it is wrong to say male and female are disjointed, but RDFS can cater for subclass relationships only; e.g. male is a subclass of human being. On other hand, it is impossible to combine or create classes using Boolean relations. The expression of many restrictions is limited. The need consequently arose for a new language to overcome all these deficiencies.

There are also many limitations to RDFS, among which are its inability to express equality and inequality, and its limited ability to define the enumeration of property

values. Regarding the latter, it cannot describe some relations among entities, such as union, intersection, unique, symmetric, transitive and inverse, and, as far as complement constraints go, it cannot apply cardinality and existence. Domain and range can only be specified globally. As a result, several languages, such as OWL and DAML+OIL, have been developed to meet these limitations.

2.5.2.3 Annotated DAML+OIL Ontology Markup

DARPA Agent Markup Language (DAML) + Ontology Inference Layer (OIL), or DAML+OIL [34, 63] is a semantic markup language designed for use in Web resources. In fact, it was built on RDF and RDFS, which is to say that it has an RDF/XML syntax based on the frame paradigm; so DAML+OIL could be considered a specific kind of RDF extended with richer modelling primitives to cope with the weaknesses of RDF /RDFS. To this end, it uses URIs to define the resources as RDF. DAML+OIL was actually developed to describe the structure of a domain, as most web-based languages describe structure in terms of classes and properties. DAML+OIL uses description logic (DL)-style model theory to formalise the meaning of a language [56].

Researchers first created OIL and a further effort produced DAML+OIL, an amalgamation of an American proposal and a European language.

2.5.2.4 Ontology Interchange Language (OIL)

A semantic markup language for Web semantics has been built on RDF and RDF/S, this language providing modelling primitives used in frame-based and DL-oriented ontologies [31].

DAML+OIL has many limitations [10]: it lacks property constructors, it has no composition or transitive closure, its only property types are transitive and symmetrical, sets are the only collection type (there are no bags or lists), there is no comparison in data value, it allows only unary and binary relations, and there are neither default values nor variables.

2.5.2.5 Web Ontology Language (OWL)

OWL [1, 18, 61, 77,83], which is a language for processing web information, became a W3C recommendation in February 2004 and was built using RDF to remedy the weaknesses in RDF/S and DAML+OIL. It provides more affluent integration and interoperability of data between communities and domains.

It can be said that there is a similarity between OWL and RDF, but the former has a stronger syntax with more machine interpretability and vocabulary language than the latter. Obviously, RDF is commonly limited to binary ground predicates, and RDFS also has the limitation that it represents a subclass hierarchy and a property hierarchy, with the domain and range definitions of these properties. In other words, the language of OWL is more expressive than that of RDF(S).

To cope with the limitations of RDF, RDFS and DAML+OIL, W3C defined OWL. Indeed, OWL is an extension of RDFS; in other words, it builds on RDF and RDFS, using XML syntax; overall, OWL uses the RDF meaning of classes and properties. W3C classifies OWL into three sublanguages, each of which is intended to supply different aspects of these incompatibilities. These are OWL Lite, OWL DL and OWL Full [76].

OWL Lite is the simplest version of OWL and provides a classification hierarchy and simple constraints; it permits only the expression of relationships with maximum cardinality equal to 0 or 1, thus being designed for easy implementation. In this sublanguage, there is some restriction of OWL DL to a subset of language constructors, with some limitations such as an absence of explicit negation or union. The disadvantage of this sublanguage is restricted expressiveness.

OWL DL is so called because it uses Description Logic to represent the relations between objects and their properties. Indeed, it provides maximum expressiveness while preserving the completeness of computational properties. OWL Lite is a sublanguage of OWL DL.

The sublanguage OWL Full provides highest expressiveness and the syntactic freedom of RDF but without preserving guarantees on computational complexity. OWL Lite and OWL DL are sublanguages of OWL Full.

Table 2.1 shows a comparison between the ontology languages discussed above.

The following table shows the limitations and differences between RDF, DMAL+OIL and OWL. The table shows many limitations to RDFS, among which are its inability to express equality and inequality, and its limited ability to define enumeration of property values. DAML+OIL has many limitations, such as property constructors; it has no composition or transitive closure, in property types contain transitive and symmetrical.

Table 2.1: Comparison between ontology languages

The expression	RDF/RDFS	DAML+OIL	OWL
Class	√	√	√
rdf:Property	√	√	√
rdfs:subClassOf	√	√	√
rdfs:subPropertyOf	√	√	√
rdfs:domain	√	√	√
rdfs:range	√	√	√
Individual	×	√	√
sameClassAs	×	√	√
samePropertyAs	×	√	√
sameIndividualAs	×	√	√
differentIndividualFrom	×	√	√
inverseOf	×	√	√
TransitiveProperty	×	√	√
SymmetricProper	×	√	√
FunctionalProperty	×	√	√
InverseFunctionalProperty	×	√	√
allValuesFrom	×	toClass	√
someValuesFrom	×	hasClass	√
minCardinality	√	√	√
maxCardinality	√	√	√
cardinality	√	√	√
oneOf	√	×	√
disjointWith	×	√	√
complementOf	×	√	√

2.6 Representation Knowledge Structures

There are many types of representation knowledge structures, including DL (ontology languages), network-based forms (semantic networks), frame-based and object-oriented representations.

2.6.1 Network-based Representation Structures

The term "network-based" [2, 3] refers to the structure of this type, so the present proposal will deal with a general view of networks containing the two important elements of nodes and links. In general, nodes are used to illustrate concepts such as classes or individual objects. Concepts could have attributes which are linked to the corresponding nodes; links are used to describe relationships between concepts.

2.6.2 Frame-based Structures

Frame-based structures [2, 3] provide modelling support to the ontology developer and user, but have limitations in that they lack a well-defined semantic. There is, however, a frame-based language called Frame Logic (F-Logic), grounded in first-order logic, which supports a well-defined and well-understood semantic. The lack of well-defined semantics is a universal problem when using a frame-based structure as an ontology language. Well-defined semantics are essential to enable applications to "understand" the ontology, or at least process it according to well-defined rules.

2.6.3 Description Logics

Description logics [2, 3, 35, 49] are a family of decidable logics for formal knowledge representation. Moreover, they are formal languages for representing knowledge used to express the conceptualisation of such domains in an organised and formally well-understood manner, leading to the provision of decidable formal semantics. They can represent concepts and their relationships (roles), also giving them formal semantics. Actually, the core descriptive tools of a DL are represented,

combining concepts in a suitable form. Thus, it is possible to describe the concepts and their relations.

This kind of representation language refers to two elements: concept description, used to describe a domain, and logic-based semantics. It is considered to define a subset of First Order Logic (FOL).

The main structures of this type are:

- Syntax (formula), which concerns the definition of how to write correct sentences in the language. In this type, a set of symbols called "concept" and a set of binary relations called "role" are defined.

- Semantics, which concerns the definition of the meaning of sentences by interpreting concepts and roles as sets of individuals. This refers to truth value and interpretation.

- Logical inference (reasoning), which derives results logically, should be decidable and efficient.

DL is considered an extension of those frame-based and semantic networks which are not capable of logic-based semantics; it is therefore also called "terminological logics".

The structure of knowledge in DL is represented in a hierarchical organisation of classes. These concepts are defined by using descriptions which identify the properties that objects must satisfy in order to belong to the concept.

Actually, a description logic theory consists of statements about concepts, individuals and their relations. Thus, DL systems allow the representation of ontologies in three components:

- ***Concepts*** in DL are as in the frame paradigm: they represent classes of objects. In other words, they are a set of individuals of the application domain that have some common characteristics, as can be the case for *people* or *cars*.

- ***Roles*** are the logical representations of relationships between concepts; for example, the role *hasFather* or the role *madeOf*. In other words, they describe binary relations between concepts; consequently they also allow the description of properties of concepts.

- ***Individuals*** represent instances of classes.

In DL, there are two special concepts, named Top and Bottom; Top (⊤) is a concept that contains all the individuals of the domain, while Bottom (⊥) is the empty concept, which also represents the contradiction.

2.6.3.1 Description Logic Families

Section describes the constructors of the DLs and, as a consequence, identifies the most common DL families. For every DL, the concepts Top and Bottom are interpreted as follows:

$$\top \equiv A \sqcup \neg A \Rightarrow \top^I = \Delta^I$$

$$\bot \equiv A \sqcap \neg A \Rightarrow \bot^I = \emptyset$$

Where: Δ^I is a interpretation domain and I is a interpretation function

Table 2.2 shows the main construct and syntax of each DL language. For example, the language SHIQ contains all constructs and syntax of the S, H, I and Q languages [2].

Table 2.2: Main constructs and syntax of DL languages [2]

Construct	Syntax	Languages			
Concept	A	FL_0	FL^-	AL	S
Role name	R				
Intersection	$C \cap D$				
Value Restriction	$\forall R.C$				
Limited existential Quantification	$\exists R$				
Top Or Universal	$\top$				
Bottom	$\bot$				
Atomic Negation	$\neg A$				
Negation	$\neg C$	C			
Union	$C \cup D$	U			
Existential Restriction	$\exists R.C$	E			
Number Restrictions	$(\geq n\,R)\,(\leq n\,R)$	N			S +
Nominals	$\{a_1 \ldots a_n\}$	O			
Role Hierarchy	$R \subseteq S$	H			
Inverse Role	$R-$	I			
Qualified Number Restriction	$(\geq n\,R.C)\,(\leq n\,R.C)$	Q			

2.6.3.2 Types of Description Logic

2.6.3.2.1 SHOIN (D) Description Logic

SHOIN (D) [35] is a type of description logic that provides a high level of expressivity and offers full negation, disjunction within verse roles and a restricted form of the universal form of existential quantification; it is therefore called “concept description”. SHOIN (D) additionally supports reasoning with concrete data-types. At present, OWL DL is correspondent to SHOIN (D). It clears that OWL DL adds very little in expressiveness to OWL Lite.

2.6.3.2.2 SHIQ (D) Description Logic

An associated logic, SHIQ (D) [35] is distinguished from SHOIN (D) essentially by not supporting nominal concepts (or named objects), allowing a qualified number of restrictions of the concept and simple roles. There is a mapping or translation from DAML+OIL to the SHIQ (D) language.

2.6.3.2.3 SHIF (D) Description Logic

SHIF (D) [35] is just SHOIN (D) with the exclusion of the *oneof* constructor and the inclusion of the (at-least) and (at-most) constructors limited to 0 and 1. In fact, OWL Lite can be translated to SHIF (D) to allow for reasoning. On the other hand, OWL Full is not computationally decidable.

2.6.4 Base of Ontology Language

Indeed, since saying that description logic is the basis of most ontology language, it is appropriate to explain briefly the base of the description to understand the ontology language clearly. OWL has been chosen because it is almost the last version of these languages, and our research is concerned with this type of RDF.

In particular, description logic named SHOIN ***(Dn)*** is used because it is the underlying logic of OWL.

The expressiveness of DLs is limited by the lack of variables. However, this limitation does the following: it ensures decidability, improves tractability and allows for efficient reasoning.

On the other hand, there are some advantages in using DL languages to build ontology. These include the fact that they are well-understood declarative semantics and that there are tried and tested algorithms to verify the completeness and decidability of a number of properties. Many relationships exist between concepts in Description Logics, including subsumption; that is, when a concept X subsumes another concept Y, meaning that X is the more general concept.

Description logics are seen as tools that support the Semantic Web and help to realise its vision. The Semantic Web uses DLs to define, maintain and integrate ontologies. One of the significant uses of ontology is to provide a common understanding of terms between different agents, which is to say to establish a joint terminology between the agents.

DLs have intuitive semantics and syntax which help to communicate the intended meaning, which is why they play a major role in several application areas such as natural language processing and database management. For DL languages to be able to model several application domains, some important features, such as effectiveness and ease of comprehensibility, should appear in all of them.

Table 2.3 is in three parts, listing the main component of ontology (i.e. class, property and individual) in OWL and DL syntaxes. These tables show how these syntaxes are built on each other in order to describe the feature of the ontology in an accepted way [2].

Table 2.3: Main Components of OWL and DL syntax [2]

OWL Syntax (Axioms of Class)	DL Syntax
Class(A partial $C_1 \dots C_n$) Or Class (A)	$A \sqsubseteq C_1 \sqcap \dots C_n$ A
Class (owl: Thing)	T
Class (owl: Nothing)	$\bot$
Class(A complete $C_1 \dots C_n$) Or IntersectionOf (C_1 C_2 ...)	$A \equiv C_1 \sqcap \dots C_n$ $C_1 \sqcap C_2$
EnumeratedClass (A $a_1 \dots a_n$) Or OneOf (a_1 a_2 ...)	$A \equiv \{a_1\} \sqcup \dots \sqcup \{a_n\}$ $\{a_1\} \sqcup \{a_n\}$
UnionOf(C1 C2 ...)	$C_1 \sqcup C_2$
ComplementOf(C)	$\neg C$
SubClassOf(C D)	$C \sqsubseteq D$
EquivalentClasses($C_1 \dots C_n$)	$C_1 \equiv \dots \equiv C_n$
DisjointClasess($C_1 \dots C_n$)	$C_i \sqsubseteq \neg C_j, (1 \leq i < j \leq n)$
Restriction	
Restriction(r SomeValueFrom(C))	$\exists r.C$
Restriction(r AllValueFrom(C))	$\forall r.C$
Restriction(r HasValue((a))	$\exists r.\{a\}$
Restriction(r MinCardinality(n))	$\geq n r$
Restriction(r MaxCardinality(n))	$\leq n r$

OWL Syntax (Axioms of Property)	DL Syntax
ObjectProperty(r super(r_1) ... super(r_n)	$r \sqsubseteq r_1 \sqcap \dots \sqcap r_n$
domain(C_1) ... domain (C_n)	$\exists r.\ T \sqsubseteq C_1 \sqcap \dots \sqcap C_n$
range(C_1) ... range (C_n)	$T \sqsubseteq \forall r.C_1 \sqcap \dots \sqcap \forall r.C_n$
[InverseOf(s)]	$r \equiv s -$
[Symmetric]	$r \equiv r -$
[Functional]	$T \sqsubseteq \leq 1\, r$
[InverseFunctional]	$T \sqsubseteq \leq 1\, r -$
[Transitive])	Trans(r)
SubProperty of (r s)	$r \sqsubseteq s$
EquivalentProperties($r_1 \dots r_n$)	$r_1 \equiv \dots \equiv r_n$

OWL Syntax (Axioms of Individual)	DL Syntax
Individual (a type(C_1) ... type(C_n)	$C_1 \sqcap \dots \sqcap C_n\,(a)$
Value(r_1 a_1) ... value (r_n a_n)	$r_1\,(a, a_1), \dots, r_n\,(a, a_n)$
SameIndividuals ($a_1 \dots a_n$)	$a_1 \equiv \dots \equiv a_n$
DifferentIndividuals($a_1 \dots a_n$)	$a_i \neq a_j, (1 \leq i < j \leq n)$

2.7 Knowledge Bases and KRS

A Knowledge Base (KB) [3] is a particular and evolved form of information system, which can contain many different conceptualizations of different domains, named ontologies, and which is managed through a Knowledge Representation System (KRS) that gives the facilities for managing and querying the KB, defining new concepts and roles and inferring new knowledge. If a KRS can only query, the KB is called a knowledge inference system or simply a reasoner. A KB is usually constituted by two elements: KB = ⟨Tbox, Abox ⟩.

A DL knowledge base is generally composed of an intentional component (TBox), which defines the concepts and roles, and an extensional one (ABox), which defines the membership of individuals and couples to concepts of individual relationships.

TBox: The TBox (terminological box) contains all the concept and role definitions, as well as the axioms of our logical theory (e.g. *"A father is a man with a child"*). The axioms of a TBox can be divided into *definitions* (C ≡ D) and *subsumptions* (C ⊑ D) ; the former is used to say that a concept C is equivalent to another concept D (atomic or complex); while the latter is used to say that a concept C is a subclass of the concept D.

The TBox contains intentional (terminological) knowledge in the form of a terminology; in other words, it contains the definitions of concepts and roles. Moreover, it is built through declarations that describe general properties of concepts.

ABox: The ABox contains all the assertions (also known as facts) of the logic theory, and an assertion is used to express a property of an individual of the domain (for example *"Tom is a father"* is represented as *Father [Tom]*). An assertion is also *R (a, b)* where *R* is a role (e.g. *hasFather (James, Tom)*). The ABox contains extensional (assertional) knowledge, which is definite and specific to the individuals (instances) of the discourse domain [3].

2.8 Reasoning

An ontology language needs to be based on a logical form to enable inferencing and reasoning. DLs consider a decidable subset of first-order logic with well-understood inference rules, but FOL is not decidable; this decidability is very convenient for reasoning about ontologies. Logics-based language is in fact required to facilitate inferencing and reasoning. Reasoning [6, 68] is used in several development phases to ensure the quality of an ontology. It could be used to check whether concepts are consistent and to obtain implied relations during ontology design. Inference engines such as FaCT and RACER have been successfully implemented in practical reasoning systems to provide reasoning services for DL.

2.8.1 Reasoning Services

Reasoning services are the tasks of the KRS [2], which can be differentiated into services for the TBox and services for the ABox [3, 49].

The services that involve only the **TBox** are:

- **Subsumption:** This task verifies whether a concept C is subsumed by another concept D(TBox $\models C \sqsubseteq D$); this is true if and only if for all the interpretations I there is($C_i^I \sqsubseteq D_i^I$).

- **Consistency:** This task verifies that there exists at least one interpretation I for a given TBox(TBox $\not\models \perp$).

- **Local Satisfiability**: This task verifies for a given concept C that there exists at least one interpretation in which $C^I \neq \varnothing$

The services for the **ABox** are:

- **Consistency:** This task verifies that an ABox is consistent with respect to a given TBox (KB $\not\models \perp$).

- **Instance Checking:** This task verifies whether a given individual x belongs to a particular concept $C(ABox \models C(x))$.

- **Instance Retrieval:** This task returns the extension of a given concept C, that is, the set of individuals belonging to C.

2.9 Summary

The goal of an ontology is to achieve a common and shared knowledge that can be transmitted between people and between application systems. Thus, ontologies play a key role in achieving interoperability across organizations, because they aim to capture domain knowledge and their role is to create semantics explicitly in a generic way, providing the basis for agreement within a domain. Thus, ontologies have become a popular research topic in many communities. In fact, ontology is a main component of our research; therefore, the definition, structure and the main operations and applications of ontology are provided.

Ontology language is the basis of ontological knowledge systems, the definition of a system of knowledge representation language specification; it not only has a rich and intuitive ability to express and use it, but the body should be easily understood by the computer, processing and applications. Thus, a brief survey of state-of-the-art ontology languages which are used to express ontology over the Web is provided; all relevant terms were shown in order to provide a basic understanding of ontologies and of description logics, which are the basis of ontology languages.

Chapter 3

Ontology Matching Techniques and Alignment Systems

3.1 Introduction

This chapter elaborates the different kinds of mismatch that can occur between ontologies. It also presents various matching methods. Finally, it provides a survey of state-of-the-art ontology alignment tools.

The Semantic Web uses ontologies to describe the semantics of the data. In other words, ontologies provide semantics for annotations in Web pages. For this reason, many ontologies have recently been developed for the WWW. Because such a large number of ontologies covered overlapping domains, the need arose for mapping between them to provide a common layer or bridge in order to exchange information. The development of new forms of existing ontologies occurred through the merging or integration of two ontologies, and by reuse through mapping or alignment of ontologies. Consequently, many tools or systems have been developed to deal with this problem.

Several ontologies have already been developed and made publicly available, and several applications are needed to access or use them. In fact, a single ontology is no longer enough to support all tasks. Hence, mapping between these ontologies has become necessary to provide a common layer by which information can be exchanged in a semantically sound manner.

3.2 Mismatching between Ontologies

Research has dealt extensively with the probability of mismatching that could appear between ontologies when trying to apply mapping [7, 41]. The mismatching between

concepts in different ontologies when mapping between them can occur at different levels.

The first of these is at the level of language, and consists of differences in encoding representations of ontology; that is to say, different languages are used to represent an ontology and the meaning of language constructs. The second is at the level of ontology; this contains mismatches in the meaning or encoding of concepts.

3.2.1 Language Level Mismatches

This type of mismatching, which can affect syntax, logical representation, semantics of primitives and language expressiveness, occurs especially when trying to apply mapping between two ontologies that have been written in different representations of ontology languages.

- **Syntax:** This kind of mismatching arises because of the different structures of language used to build the ontology. Different ontology modelling languages use different syntaxes to build the components of ontology, e.g. <rdfs: Class ID = “Chair”> in RDF Schema vs. (defconceptChair) in LOOM.

- **Language Expressiveness or Expressive Power**: Some languages are able to represent expressions or objects that cannot be represented in others.

- **Semantics of Primitives:** This happens when different languages have the same components in two languages, but with different semantics.

- **Logical Representation:** This mismatching level concerns the difference in representation of logical notions.

3.2.2 Ontology Level Mismatches

This type of mismatch may occur in many different cases, including instances where ontologies are written in the same language, as well as when they use different ones.

3.2.2.1 Content Level

- **Scope:** When two terms have the same meaning, but with different instances.
- **Model Coverage and Granularity:** This type could involve a mismatch in the style of modelling or terminological mismatches.

3.2.2.2 Organisation Level

- **Synonym Term** / multi–language mismatches occur when using different synonyms for the same concept, as in "car" and "automobile".
- **Homonym Terms** mismatch different meanings for the same concept; this is also called overlapping terminology. For instance, "table" can be a piece of furniture or a list of information arranged in columns and rows.
- **Concept Structuring** concerns the difference in the design of ontologies: one might use the concept of "address" while another disperses the addresses throughout many fields.
- **Encoding** mismatches happen when ontologies are built in different formats such as dollars and pounds; in this situation the conversion function should be used, e.g. a date represented as dd/mm/yyyy or mm-dd-yy.
- **Paradigm** mismatches occur when more than one paradigm is used to represent a concept, e.g. one model uses temporal representations based on interval logic, while another uses a representation based on points.

3.3 Basic Matching Strategies

Ontology matching is carried out in order to specify matching relations among concepts from different ontologies. Such a rich set of semantic relations for expressing alignment is useful in ranking. This matching relies on similarity measures to establish alignment. In other word, ontology matching or alignment is the process of finding the closest semantic and intrinsic relationship between the existing

ontologies of corresponding ontological entities. Therefore, this section describes several similarity measures and matching strategies.

3.3.1 General Properties of Similarity

There are many methods which may be used to measure the similarity between two entities. The term 'ontology similarity', as used in this work, refers to the comparison of whole ontologies or sub-elements thereof. This comparison returns a numerical value indicating whether the two elements have a high or low degree of similarity. This can be formally laid down through the following definition [28].

Similarity σ: $O \times O \rightarrow R$ is a function used to express the similarity between two entities such that [28]:

- $\forall x, y \in O, \sigma(x, y) \geq 0$ (positiveness)
- $\forall x \in O, \forall y, z \in O, \sigma(x, x) \geq \sigma(y, z)$ (maximality)
- $\forall x, y \in O, \sigma(x, y) = \sigma(y, x)$ (symmetry)

Dissimilarity is defined as follows.

Given a set O of entities, a dissimilarity δ: $O \times O \rightarrow R$ is a function from a pair of entities to a real number such that [28]:

- $\forall x, y \in O, \delta(x, y) \geq 0$ (positiveness)
- $\forall x \in O, \delta(x, x) = 0$ (minimality)
- $\forall x, y \in O, \delta(x, y) = \delta(y, x)$ (symmetry)

A distance (or metric) δ: $O \times O \rightarrow R$ is a dissimilarity function satisfying definiteness and triangular inequality:

- $\forall x, y \in O, \delta(x, y) = 0$ if and only if $x = y$ (definiteness)
- $\forall x, y, z \in O, \delta(x, y) + \delta(y, z) \geq \delta(x, z)$ (triangular inequality).

3.3.2 String-based Methods

String-based methods [28] take advantage of the structure of the string; for that reason, they are based on the structure of the string (as a sequence of letters). Therefore, string-based methods, in general, will typically find classes like Book and Textbook to be similar, but not classes Human and Person. A number of similarity measurement techniques, such as the cosine similarity measure [67, 75], Dice's coefficient [37] and Jaccard's index [92], have been defined to compute this similarity. Several algorithms have already been created to compare labels, e.g. the edit distance. There are several techniques to compare strings, depending on the way the string is viewed (as a set of letters, a set of words, etc). Therefore, the entities are probably identical if their labels are the same. In order to find a similarity amongst the names, the labels or the comments of entities, terminological methods can be applied to compare strings. These can be used for comparing class names and/or URIs [92]. The most frequently used methods are presented below.

String equality is defined as a similarity

$\sigma: S \times S \rightarrow [0\ 1]$ such that $\forall x, y \in S, \sigma(x, x) = 1$ and if $x \neq y, \sigma(x, y) = 0$.

String equality returns "0" if the strings under consideration are not identical and "1" if they are identical. This can be taken as a similarity measure. In general, it can be performed after some syntactic normalisation of the string, e.g. downcasing.

The similarity model can be divided into single or multiple [45]. The single model is based on a single measure, such as the Jaccard coefficient [88]. The multi-similarity model aggregates two or more similarity measures, such as similarity flooding [65], and hybrid measures, which can be combined like linguistic and structural similarity measures, by decreasing the number of passes over the ontology; this kind of matching can present better performance than the execution of multiple matchers.

Normally, the name or label of a class is explained as a sequence of letters from an alphabet: a chain of characters, a string in some natural language, without blank characters (space). A name of class may be a word, a term or a phrase. In identifying

the class, this name should be unique in the ontology [28]. The similarity calculation of two class names is performed in two steps:

- Normalization techniques, which are used for reducing strings to be compared with a common format.
- The comparison technique, which is a similarity method.

Many string-based techniques, such as prefix, suffix, edit, and n-gram distances [1], have been extensively used in matching systems. There are also different techniques used to compare strings based on the way the string is viewed: for example, the following is the most frequently used method:

- ***Prefix:*** takes two strings as input then tries to check whether the first string starts with the second string. For example, net = network: they are equal; but also hot = hotel [28, 38].
- ***Suffix:*** takes two strings as input then tries to check whether the first string ends with the second string. For example, phone = telephone; but also man = human [28, 38].
- ***N-gram*** takes two strings as input and computes the number of the same n-grams (i.e., sequences of n characters) between them. For example, trigrams (3) for the word *address* are *add*, *ddr*, *dre*, *res*, *ess*. [28, 38].
- ***Jaro Module*** [15, 38] is a string similarity metric used to calculate the distance between two terms rapidly with a specific metric. It is based on the number of characters which are common to the two strings. This measure is not based on an edit distance model and is not a similarity measure, because it is not symmetric. It has been defined for matching proper names that may contain similar spelling mistakes.

The Jaro measure is a non-symmetric measure σ: $S \times S \rightarrow [0, 1]$ such that

$$\text{Jaro}(s1, s2) = \frac{1}{3}\left(\frac{|s1'|}{|s1|} + \frac{|s2'|}{|s2|} + \frac{|s1'| + \text{Tp}_{s1',s2'}}{2|s1'|}\right)$$

Where s1, s2: input strings; |s1'|: number of characters in s1 that are common with s2; |s2'|: number of characters in s2 that are common with s1; $\text{Tp}_{s1',\ s2'}$: number of transpositions of characters in s1' relative to s2'.

3.3.2.1 Edit Distance

Naturally, an edit distance is the minimal cost of the best sequence of edit operations to be applied to s in order to obtain t. Naturally, it takes as input two strings and calculates the number of the operations that are usually considered, such as insertions, deletions and substitutions (replacement of a character by another) of characters required to transform one string into another.

Given a set Opr of string operations (Opr: S → S), and a cost function w: Opr → R, where a smaller value of R indicates greater similarity between strings, each operation must be assigned a cost; finally, the distance between two strings is the sum of the less costly set t of each operation.

$$\delta(s1, s2) = \min_{(opri)I;\, oprn(\ldots opr1(s1)) = s2} \left(\sum_{i \in I} w_{opri}\right)$$

The edit distance is a dissimilarity δ: S × S → [0 1], where δ (s1, s2) is the cost of the less costly sequence of operations which transforms s1 into s2 [13].

3.3.2.1.1 Levenshtein Module

The Levenshtein distance [58] is an algorithm based on the *edit-distance function*. In fact, it is based on the consideration that the distance is the number of operations necessary to convert the first string into the second one. These operations are insertion, deletion and substitution, where cost is 1.

There are many uses for the Levenshtein algorithm, for example:

- Spell checking.
- Speech recognition.
- DNA analysis.
- Plagiarism detection.

Other examples of such measures are the Smith–Waterman measure [43] and the Gotoh [43] and Monge-Elkan [70] distance functions.

3.3.2.2 Token-based Distances

This technique considers a string as a (multi)set of words (also called bag of words) or as a vector $\vec{}_s$ belonging to a metric space *V* in which each dimension is a term (or token) and each position in the vector is the number of occurrences of the token in the corresponding bag of words. These approaches are used for comparing pieces of texts rather than labels, therefore, they usually work well on long texts; moreover, it is helpful to take advantage of other strings that are attached to ontology entities. For this reason, the Term Frequency/Inverse Document Frequency (TF/IDF) is used in our system to calculate similarity between descriptions of the concepts or relations.

3.3.2.2.1 Term Frequency/Inverse Document Frequency

Many systems use measures based on TF/IDF, so it is considered to be a universal measure [80] which is widely used in the information community and for scoring the relevance of a document, i.e., a bag of words, to a term by taking into account the frequency of appearance of the term in the corpus. In other words, these measures compute, for each term in the strings, their relevance with regard to the corpus, based on TF/IDF. After that, they use vector space techniques to compute a distance between the two strings. Actually, this does not measure similarity, but assesses the relevance of a term to a document.

TF/IDF is defined as:

$$V'(w, S1) = \log(TF_{w,S1} + 1) * \log(IDF_w)$$

Where $TF_{w, S1}$ is the frequency of word w in S1; IDF_w is the inverse of the fraction names in the corpus that contain w.

$$TF - IDF(S1, T) = \sum_{\omega \epsilon S \cap T} V(\omega, S1) \cdot V(\omega, T)$$

$$V(\omega, S1) = \frac{V'(\omega, S1)}{\sqrt{\sum_{\omega'} V'(\omega, S1)^2}}$$

3.3.2.2.2 Jaccard Similarity

Jaccard similarity is measure used to compute the similarity based on the probabilistic interpretation of the set of instances [51].

Given two sets S1 and S2, let P(X) be the probability of a random instance occurring in the set X. The Jaccard similarity is defined as follows:

$$Jaccard\_sim(S1, S2) = \frac{P(S1 \cap S2)}{P(S1 \cup S2)}$$

This measure is normalised and obtains 0 when there is no instances in common S1 ∩ S2 = ∅ and 1 when S1 = S2. Moreover, it can be used with two classes of different ontologies sharing the same set of instances.

Therefore, based on the matching of instances, the general way to compare classes when they share instances is to check the intersection of their instance sets S1 and S2. Then the results will based on the set relations; for example, these classes will be very similar if S1 ∩ S2 = S1 = S2, more general when S1 ∩ S2 = S2 or S1, or disjoint when S1 ∩ S2 = ∅.

3.3.3 Language-based Methods

Language-based [28] methods rely on using natural language processing techniques to help extract the meaningful terms from a text. Terms are phrases that identify concepts; they are often used for labelling concepts in ontologies. Comparing these terms and their relations should help to assess the similarity of the ontology entities they name and comment upon. These are based on linguistic knowledge; indeed, there are two general techniques, one relying on algorithms only, while the other can use external lexicon-based resources such as dictionaries. As a consequence, ontology matching can take great advantage of recognising and identifying them in strings; in other words, it uses the internal linguistic properties of the instances, such as their syntactic properties.

In general, extrinsic linguistic methods [28] are used for finding extra similarities between terms; external linguistic resources such as dictionaries and lexicons increase the chances of finding matching terms. Lexical information (e.g. names, definitions and distance between strings) can help with reclassification of matching results. Auxiliary information (e.g. WordNet) provides semantics for the elements in ontologies.

- **A lexicon**, or dictionary, is a set of words together with a natural language definition of these words. Of course, for a particular word, e.g. ‘article’, there can be several such definitions. Dictionaries can be used with gloss-based distances.

- **A terminology** is a kind of lexicon, but it contains phrases rather than single words. It is usually built for a specific domain, so it is considered less equivalent than a dictionary.

- **Thesauri** can improve the similarity measure. In general, there are two relations in thesauri: noun and verb relations. A thesaurus is a kind of lexicon with more relational information added, for example:

- Hypernyms are words used to name different entities; e.g. ‘biography’ is a more general term than ‘autobiography’.

- Synonyms are different words used to name the same entity; e.g. ‘paper’ can mean the same as ‘article’.

- Antonym, e.g. ‘practice’ is the opposite of ‘theory’.

These are examples of extrinsic methods based on WordNet [67, 66], an electronic thesaurus database for English which distinguishes clearly between word senses by grouping words into *synsets* (concepts or senses of groups of terms) or sets of synonyms. A synset is a set of synonyms denoting the same concept, paired with a description (or gloss) of the synset. For instance, the words ‘night’, ‘night-time’ and ‘dark’ constitute a single synset that has the following gloss: the time after sunset and before sunrise while it is dark outside. Synsets are interconnected through explicit semantic relations, some of which (hypernymy [is-a-kind-of] and hyponymy for nouns, and hypernymy and troponymy for verbs) constitute kind-of (or is-a) and part-of (holonymy and meronymy [is-a-part-of] for nouns) hierarchies. For example, a tree is a kind of plant, so ‘tree’ is a hyponym of ‘plant’ and ‘plant’ is a hypernym of ‘tree’.

WordNet consists of such sets of synonyms, which provide different interrelationships such as:

- Synonymy: using different terms for the same concept (car →automobile).

- Antonymy: using a term opposite in meaning (man ↔ woman).

- Hypernymy: using a more general term from another (dinner →meal).

The WordNet::Similarity package is a collection of tools for measuring semantic similarity. The five measures used here based on the structure and content of WordNet. These measures are the Leacock-Chodorow measure [53], the Jiang-Conrath measure [79, 87], the Lin measure [79, 87], the Resnik measure [79, 87] and the Wu-Palmer measure [79, 87]. These measure the semantic similarity between a pair of concepts. They use information found in an *is–a* hierarchy of concepts, taking

as input two concepts and returning a numeric value that represents how much concept A is like (or similar to) concept B. These measures can be divided into

1. measures are based on the notion of information content like The Lin, Resnik and Jiang-Conrath, and,

2. Measures are based on path length like the Leacock-Chodorow and Wu-Palmer.

Before one starts matching, one has to apply some linguistic normalisation operations, which aim to reduce each form of a term to some standardized form that can be easily recognised [28].

In general, the tag name can consist of characters such as '*', '/', '-' and other letters; these are treated as "noise" and have to remove them from the terms before strictly comparing strings which have meaning in natural language. In order to remove them, there are several normalization procedures which help in improving the results of subsequent comparison and removing ambiguity; these include the following:

- **Case Normalisation** is used to convert each alphabetic character in the strings into lower case; for example, "BOOK" becomes "book".;

- **Blank Normalisation** consists of normalising all blank characters into a single blank character;

- **Link Stripping** consists of normalizing some links between words, it is used to replace apostrophes and blank underline with dashes or blanks.

- **Punctuation Elimination** suppresses punctuation signs; for example, "C.D." becomes "CD";

- **Stopword Elimination** eliminates tokens such as articles, prepositions and conjunctions (common words such as 'a', 'to', 'in').

3.3.4 Structural Methods

In this method of matching, instead of comparing their names or identifiers, the structures of entities that can be found in an ontology are compared. In this kind of matcher, information is used about the structure, such as subclass and super-class relationships, domain and range of properties, and the graph structure of ontologies. In fact, this information provides insight into ontologies. This kind of comparison can be divided into the following:

- **Internal Structure:** this method is comparing the internal structure of entities, in other the words, the similarities between the names of their properties (e.g., the value range or cardinality of their attributes).

- **External Structure** this method is comparing the relations of the classes with other classes like compute the similarity the super-classes of the two classes.

3.3.4.1 Taxonomic Structure

Taxonomic structure [74] has been studied in detail by researchers, because it is considered to be the backbone of ontologies. In fact, a taxonomy is a graph made with the subClassOf ("is-a") relation. For this reason, it is very often used as a comparison source for matching classes.

In a taxonomy, a concept defined by the label (a term which may consist of several words) is associated by subsumption relations which link it to other concepts. A taxonomy is a set of concepts linked by is-a relations, represented by acyclic graphs. The concepts are represented by node-connected graphs directed by the links corresponding to is-a relations. Given two taxonomies, the aim is to match the elements of one, called the taxonomy source, with the elements of the other, called the taxonomy target. Matching is assumed to be determined by relations of the 1:1 type. The alignment process is designed to generate two types of relation: equivalence and specialization.

3.3.5 Matching Based on Instances

In a case where two ontologies have similar instances, finding corresponding concepts based on checking similarities between the individuals is required. If the similarity level of two instances reaches a threshold, then the two individuals can be considered as matched. For example, the name of the author of a book will not change, even if people play different roles in different ontologies. The matching can be based on instance comparisons. To identify the similarity level of two instances, string similarity methods may be used.

3.4 Composition of Matching Methods

All alignment and similarity methods use several strategies to provide similarity values that have to be aggregated in order to compose a particular algorithm. For example, computing the similarity between two classes requires the aggregation, in a single similarity measure, of the similarity obtained from their names, super-classes and properties, and that of their instances.

3.4.1 Similarity Aggregation

Indeed, once the similarities between elements from different ontologies are available, based on different strategies (e.g. string similarity, structure similarity or instances), it is necessary to aggregate similarity algorithms in order to combine matchers. Combining different similarity methods is an effective way to achieve high accuracy for a larger variety of ontologies than would be achieved by a single similarity method. For this purpose, many approaches combining the results of several independently executed mapping algorithms are proposed [23, 26, 75].

To select the match candidates, many strategies can be used; these will aggregate matcher-specific similarity values for every element pair.

Triangular Norms [28] are used as conjunction operators in uncertain calculi.

A triangular norm T is a function from D × D → D (where D is a set ordered by ≤ and provided with an upper bound T) satisfying the following conditions:

- T(a, T) = a (boundary condition)
- a ≤ b ⇒ T(a, c) ≤ T(b, c) (monotonicity)
- T(a, b) = T(b, a) (commutativity)
- T(a, T(b, c)) = T (T(a, b), c) (associativity).

Triangular norms are suitable for combining the highest score from all aggregated values; moreover, they tend to express the dependencies between the values of the different dimensions.

Typical examples of triangular norms are min(a, b), a×b and max(a+c−1, 0). All are normalised if the measures provided to them are normalised; min is the only idempotent norm (∀a, min(a, a) = a). Triangular norms are the obvious candidates for a combination that requires the highest score from all aggregated values. Owing to association, triangular norms can be extended to n-ary measures. Any triangular norm over the unit interval can be expressed as a combination of these three functions.

Another triangular norm for aggregating several dimensions is the weighted product.

Weighted Sums [28]: Weighted linear aggregation considers that some of the values to be aggregated do not have the same importance. For instance, similarity in properties is more important than similarity in comments. The aggregation function will thus use a set of weights w1, . . . ,wn corresponding to a category of entities, e.g. classes, properties. The aggregation function can be defined as follows:

Let O be a set of objects which can be analysed in n dimensions; the weighted sum between two such objects is as follows:

$$\forall s, s' \in O, \delta(s, s') = \sum_{i=1}^{k} wi \times \delta(si, si')$$

Where δ(si, si') is the dissimilarity of the pair of objects along the ith dimension and wi is the weight of dimension i.

This kind of measure can be normalised, if all values are normalised, by having:

$$\sum_{i=1}^{k} wi = 1$$

It appears that the measure on the instances is more accurate than those on the labels. This can be inferred from the fact that there are no common names in both sets of labels, or that there are lower distances in the latter case. Thus, weighting these dimensions could be promising.

Weighted Average: Fuzzy aggregation operators are used for assimilating homogeneous quantities in a way that preserves the structure of the aggregated domains.

Let O be a set of objects which can be analysed in n dimensions. The weighted average between two such objects is as follows:

$$\forall s, s' \in O, \delta(s, s') = \frac{\sum_{i=1}^{k} wi \times \delta(si, si')}{\sum_{i=1}^{k} wi}$$

such that δ(si, si') is the dissimilarity of the pair of objects along the ith dimension and wi is the weight of dimension i.

A simple average function is a function such that all weights are equal. If the values are normalised, the weighted average is normalised. In fact, the normalised weighted sum is also a weighted average.

3.5 Overview of Existing Approaches to Ontology Alignment and Mapping

Recently, many approaches, systems or tools for ontology mapping or merging have been developed, and they cope with different problem areas. Ontology mapping methods can generally be classified into two approaches: concept-based (top-down) approaches consider concept information such as name, taxonomies and relations, and properties of concept elements for ontology merging, while instance-based (bottom-up) approaches base the structural hierarchy on instances of concepts and relations. For example, in this approach are Chimaera [61], PROMPT [68] and FCA-Merge [18]. The following literature offers several approaches to the alignment of ontologies, based on measures of similarity**.**

- **IF-Map** [47] is a fully automatic method of mapping ontologies based on the Barwise-Seligman theory of information. IF-Map can support several languages, including RDF, KIF, Ontolingua, Protégé KBs and Prolog KB [47]. This method consists of four major steps: ontology harvesting, translation, infomorphism generation and display of results. In ontology harvesting, an ontology can be obtained by using various ways of importing resources, such as using existing ontologies which are available on ontology libraries, editing them or harvesting them from the Web. Translation is used to translate a variety of formats such as RDF, KIF, Ontolingua, Protege KBs and Prolog KB into Horn clauses. This method is declaratively specified in Horn logic and executed with a standard Prolog engine. Infomorphism generation begins once the translated ontology becomes ready; it is the process of mapping between ontologies. Finally, display mappings display the output in RDF format so that it can be accessed by other Semantic Web applications. For additional processing and reference, they are stored in a KB.

  Advantage and Disadvantage of IF-Map: this method applies string matching and structure matching. But it does not provide instance matching or semantic matching; moreover, it doesn't use auxiliary information. The mean of defining

ontology is based on concepts only. Moreover, it does not apply a normalisation process or any aggregation methods. The way of selecting matching elements is based on height values.

- **GLUE** [21] is a semi-automatic system that employs machine-learning techniques to find mappings. It uses multiple learning strategies to cope with different types of information, either in data instances or in the taxonomic structure of the ontologies, in order to make predictions. GLUE checks two ontologies for concepts with the greatest degree of similarity between them by using probabilistic definitions of several practical similarity measures. The establishment of the similarity of two concepts in two different ontologies is based on the sets of instances that overlap between those two concepts. GLUE's general architecture consists of three main modules: Distribution Estimator, Similarity Estimator and Relaxation Labeller. Distribution Estimator uses a set of base learners and a meta-learner approach based on a sample mapping set. It learns a strategy to identify equal instances and concepts, taking as input ontologies O1 and O2, together with their individual data, then applies machine learning to compute, for every pair of concepts A(O1) and B(O2), the four probabilities P(A;B), P(A';B), P(A;B') and P(A';B'). Thus a total of 4|O1||O2| numbers will be computed, where |Oi| is the number of concepts in ontology Oi. GLUE checks every candidate mapping. Similarity Estimator simply applies a user-supplied or defined function to compute the similarity for each pair of concepts A1 (O1), B1 (O2), based on the learnt rules that lead to the derivation of the mapping of concepts. Relaxation Labeller starts by taking as an input the similar values for the concepts of ontologies, together with domain-specific constraints and heuristic knowledge, then searches for the best mapping configuration, which will be the output. Relaxation Labelling is used to further compare concepts and relations; more meaningful results will be obtained by repeating this step and its interpretation several times. The other two steps are carried out just once.

  Advantage and Disadvantage of GLUE: this method applies machine learning techniques for alignment, moreover, it applies string matching (i.e., name only),

instance matching and semantic matching but it doesn't provide structure matching and it doesn't use auxiliary information. The mean of defining ontology is based on concepts, properties, and instances. It does not apply a normalisation process. The input for this approach is two ontologies / schemas with their data instance and the output is a set of correspondence relationship. The way of selecting matching elements is based on highest value.

- **ONION (Ontology composition)** [69] is a merging approach which was implemented by Stanford University's Database Group. It provides articulation rules for resolving terminological heterogeneity and enables knowledge interoperability that will lead to a bridging of the semantic gap between different ontologies. In other words, it is an architecture based on a sound formalism to support a scalable framework for ontology integration that uses graph-oriented models for the representation of ontologies. ONION uses both lexical and graph-based techniques to suggest articulations. The method of finding lexical similarity between concept names uses dictionaries and semantic-indexing techniques based on co-occurrence of words in a text corpus. It uses the linguistic matcher to identify every possible pair of concepts in ontologies and assigns a similarity score to each pair. It then compares the similarity score with a threshold to determine whether or not to accept it; if it does, an articulation rule is generated. After linguistic matchers are presented, a structure-based matcher starts looking for further matches. Articulation rules express the relationship between concepts belonging to the ontologies. There are several semantic relationships with a built-in meaning: (SubClassOf; PartOf; AttributeOf; InstanceOf; ValueOf).

  Advantage and Disadvantage of ONION: the inputs for this tool are terms of ontology (IDL, XML-Based) and the output is articulation rules between ontologies. It provides string, structure matching and it use dictionary but it does not provide an instance matching. The means of defining ontology is based on concepts and properties. This approach does not use a normalisation process. The way of selecting matching elements is not specific.

- **The Naive Ontology Mapping** (**NOM)** [24] approach is simple, constituting a straightforward baseline for later comparisons. It comprises six steps. Feature Engineering demands that the ontologies be represented in RDF. Search Step Selection compares all entities of the first ontology with all entities of the second. Similarity Computation computes the similarity between entities in different ontologies, using a wide range of similarity functions. In Similarity Aggregation, NOM highlights individually significant similarities by weighting individual similarity results and aggregating them. This, however, neglects individual similarities that are of less significance. Interpretation uses the individual or aggregated similarity values to derive mappings between entities. Finally, Iteration repeats the previous step several times. This gives the capacity to access the already computed pairs and use more sophisticated structural similarity measures, whereas neglecting this step provides only a comparison based on labels and string similarity. A new version has subsequently appeared with more features and heuristic combinations, such as Quick Ontology Mapping (QOM) [26].

  Advantage and Disadvantage of Naive Ontology Mapping: this approach applies string matching, structure matching and an instance matching, but it doesn't use auxiliary information. The means of defining the ontology is based on concepts, properties, and instances. The input-ontologies for this approach are in RDF format only. Moreover, it does not use a normalisation process. The way of selecting matching elements is threshold based.

- **OntoMorph** [14] is an online system for symbolic knowledge, providing a powerful rule language for specifying mappings, then facilitating ontology merging and finally rapidly generating knowledge-based translators. It combines syntactic and semantic rewriting, which are powerful mechanisms to support translation between different knowledge representation languages. Syntactic rewriting is achieved by pattern-directed rewrite rules for sentence-level transformation based on pattern matching. Semantic rewriting is done through semantic models and logical inference, and is supported by PowerLoom. OntoMorph is fully integrated within the PowerLoom knowledge representation

system, which allows transformations between different knowledge representation languages. This system does not support XML.

Advantage and Disadvantage of OntoMorph: this approach is Transformation system for symbolic knowledge. But it Transforms are expressed manually. It uses two mechanisms; Syntactic rewriting via pattern-directed rewrite rules and Semantic rewriting that modulates. In fact, it uses string matching only. This approach deals with Language mismatching level but without expressivity and deals with Ontology level mismatching but without coverage of model.

- **PROMPT** [75] is a tool for merging ontologies, developed by Stanford University Knowledge Systems Laboratory. The knowledge model underlying PROMPT is frame-based and is compatible with Open Knowledge Base Connectivity. In general, this tool provides a semi-automatic approach to merging two ontologies; it is based initially on alignment relations, which should be held before providing output as a coherent ontology. More specifically, PROMPT performs some tasks automatically: it takes two ontologies as input and creates an initial list of matches based on class names. This list will be a coherent ontology. The following cycle then occurs: (1) the user triggers an operation by either selecting one of PROMPT's suggestions from the list or by using an ontology-editing environment to specify the desired operation directly; and (2) PROMPT performs the operation, automatically executes additional changes based on the type of the operation, generates a list of suggestions for the user, based on the structure of the ontology around the arguments of the last operation, and determines conflicts that the last operation introduced in the ontology, finding possible solutions for them. PROMPT then guides the user in performing other tasks for which his intervention is required. Its top level contains Classes (collections of objects arranged into hierarchies), Slots (binary relations), Facets (ternary relations) and Instances (individual members of classes).

  Advantage and Disadvantage of PROMPT: an interactive ontology-merging tool. It applies string matching and semantic matching but it does not provide instance or structure matching. The input-ontologies for this approach are in

different format like RDF(s), OWL-Lite, and OWL-DL. The output is merged ontology. The way of defining ontology is based on concepts, properties and instances. It does not deal with normalisation process. The way of selecting matching elements is based on highest value. This approach provides interactive suggestions to the users. It solves mismatches at terminological and scope of concept level, and it helps alignment by providing possible edit points and it supports repeatability. But it is not automatic which means every step requires user interaction.

- **Chimaera** [62, 64] is a semi-automatic or interactive tool for merging ontologies. The engineer is in charge of making decisions that will affect the merging process. This tools starts by analysing the ontologies to be merged. It automatically finds linguistic match merges, and if it cannot find any matching terms, it gives the user control over any further action. In fact, it is similar to PROMPT, as both are embedded in ontology editing environments and offer the user interactive suggestions.

  Advantage and Disadvantage of Chimaera: an ontology merging tool which supports ontology browsing and editing. It uses string matching, semantic matching and structure matching but it does not provide instance matching. The input-ontologies for this approach are OKBC ontologies and the output is a merged ontology. This approach analyses ontologies to be merged; if linguistic matches are found then the merge is processed automatically; otherwise, it uses subclass and super class relationship. In fact, this approach solves mismatches at the terminological level in a very light way, and provides interactive suggestions to the users. It solves mismatches at terminological and scope of concept level, and it helps alignment by providing possible edit points and it is not repeatability. But it is not automatic which means everything requires user interaction. (It is very similar to PROMPT).

- **FCA-Merge** [86] is a method of ontology merging using linguistic analysis, based on extracted instances to derive a lattice of concepts as a structural result. This method is based on Formal Concept Analysis and lattice exploration. FCA-Merge

is a bottom-up approach, which means that for the source ontologies, it extracts instances that require merging. In other words, this method compares two ontologies that have shared instances or a shared set of documents annotated with concepts from source ontologies. FCA-Merge suggests that strong assumptions have to be met to obtain good results. The process of merging two ontologies consists of three steps: the first involves extracting instances and computing two formal contexts; the second applies the FCA-Merge core algorithm, which derives a common context and computes a concept lattice; the results of this derivation will finally generate the ultimate merged ontology. This algorithm suggests some relations, such as equivalence and Sub-Class / Super-Class.

Advantage and Disadvantage of FCA-Merge: a bottom-up approach for ontology-merging. It uses instance and structure matching. Moreover, techniques from natural language processing and formal concept analysis are applied. The inputs for this approach are a set of document of concepts but the input format is not specific and the output is a merged ontology. Input documents should be domain-dependent and each document should cover all concepts from source ontologies and must separate the concepts well enough. The way of defining ontology is based on concepts and instances. It uses a normalisation process. The way of selecting matching elements is not specified.

- **HCONE-merge** [50] has been proposed by the Human Centered Ontology Engineering Environment. The progress of HCONE-merge begins with alignment, which is done automatically by mapping ontology concepts to WordNet senses using the Latent Semantics Indexing method (LSI) to exploit linguistic and structural knowledge about ontologies and to associate concepts to their informal meaning. LSI is a vector space technique for information retrieval and indexing. The alignment stage using the reasoning services of DL to exploit the semantics of concepts by validating the mapping between concepts and finding a minimum set of axioms for the merged ontology. In the merging process, humans are involved both in capturing the proposed semantics of terms by means of informal

definitions (supported by LSI) and in the event that the relations between concepts are not stated formally, when they will need to clarify those relations.

Advantage and Disadvantage of HCONE-merge: this approach applies string matching (i.e. name only) with structure matching (i.e. parent, children) and it uses auxiliary information. The input-ontologies for this approach are in OWL-DL. The way of defining ontology is based on concepts and properties. This approach does not use a normalisation process. The way of selecting matching elements is not specific.

- **S-MATCH** [39] is a semantic matching approach between two ontologies or schemas. The main idea of this system is to take two graph-like structures (e.g. conceptual hierarchies, database schemas and ontologies) as inputs and try to establish a strong semantic relation (e.g. equivalence, more general, less general, mismatch or overlapping) between the nodes of the two graphs. Therefore, the output will be semantically corresponding relations between the nodes. The main point of applying this approach is to compute semantic matching, so to determine the relation, the meaning (not the labels) of concepts must be analysed – that is, codified in the structures of ontologies. The mapping element is a 4-tuple < IDxy, n1x, n2y, R >, x=1,..., N1;y=1,...,N2, where:

  - IDxy is a sole identifier of the given mapping element;
  - n1x is the x-th node of the first graph, n2y in the second graph;
  - N1, N2 are the numbers of nodes in the first and second graphs; and
  - R is a semantic relation that holds between the concepts of nodes n2y and n1x.

  Advantage and Disadvantage of S-MATCH: this approach applies linguistic matching (i.e., label), structure matching (i.e., path from the root) and it uses auxiliary information. The input-ontologies for this approach are not specific. The way of defining ontologies is based on concepts only. This approach uses a

normalisation process. The output is semantic relation; therefore, the way of selecting elements is not specific.

Table 3.1 compares the ten approaches to mapping discussed above.

Table 3.1: Comparison between Mapping Approaches

APPROACH	INTEROPERABILITY LANGUAGES	ONTOLOGY STRUCTURE	STRATEGY	ADDITIONAL RESOURCES	LEVEL OF AUTOMATION
IF-Map[47]	RDF,KIF, Ontologua, Protégé -KB, Prolog	Concept	Linguistic, Heuristic, Reasoning	Reference Ontology	Automatic
GLUE[21]	-	Concept, Properties, Instance	Probability	-	Automatic
ONION[69]	IDL, XML-Based	Concept, Properties	Linguistic	WordNet	Semi-Automatic
NOM[24]	RDF	Concept, Properties, Instance	Linguistic, Heuristic	Domain Specification Dictionary	Automatic
OntoMorph [14]	KIF, Loom, MELD, PowerLoom, Ontolingua	-	Reasoning, Heuristic	-	Manual
PROMPT[75]	OWL, RDFS	Concept, Properties, Instance	Linguistic, Heuristic	-	Semi-Automatic
Chimaera [62]	Ontologua		Linguistic, Heuristic	-	Semi-Automatic
FCA-Merge [86]	-	Concept, Instance	Linguistic, Heuristic	-	Semi-Automatic
H-CONE [50]	OWL-DL	Concept, Properties,	Linguistic, Reasoning	WordNet	Semi-Automatic
S-Match [39]	-	Concept	Linguistic, Reasoning	WordNet	Automatic

3.6 Summary

This chapter described the mismatching that could appear when trying to integrate, map, align or match two ontologies. It also describes many matching strategies and operations. Finally, it mentions some tools and systems that have been developed in these areas.

More generally, ontology alignment plays an important role in solving interoperability in heterogeneous systems and in many application domains.

The objective of a system of information integration is to provide a uniform view of sets of information sources on the same scope, but created independently of each other, which can be differentiated by formats, structures, modes of access or the terms represented.

Ontology Matching Methods:

The following are classifications of ontology matching methods.

Terminological Matching: This method is computing similarities based on the strings of class and property names.

- **String-based** (e.g., edit distance);
- **Semantic based** comparing the interpretations of the elements.
- **Lexicons-based**: used dictionary to determine the relation between concepts.

Internal Structure: this method is comparing the internal structure of entities (e.g., the value range or cardinality of their attributes).

External Structure this method is comparing the relations of the classes with other classes (i.e., super-classes).

Taxonomical Structure comparing the position of the entities within taxonomy;

Extensional Comparison (Instance) this method is comparing the known extension of classes, i.e. instances of classes.

Chapter 4 The Ontology Alignment Framework

This chapter illustrates in detail the main components of our system and explains how they interact with each other. It describes the multi-strategies that are used and shows how the system can aggregate different results which are produced by different strategies. Finally, it indicates how several types of format can be produced.

4.1 Overview of System

In many applications, such as the exchange of documents on the Web, several ontologies partially or totally covering the same area are involved. To enable interoperability of applications and/or agents based on these ontologies, the heterogeneity between knowledge expressed in each of them must be resolved. To this end, the semantic relationships between entities belonging to two different ontologies must be established; this is the aim of aligning ontologies.

In general, the match operation takes as input two ontologies and determines an alignment indicating that the elements of the input ontologies logically correspond to each other, i.e. they match.

Two ontologies having been given, alignment produces a set of matches, each between two entities (e.g. concepts, instances, properties, terms etc.) in terms of a relationship (equivalence, subsumption, incompatibility etc.), which may be fitted with a degree of confidence. All matches, also called alignments, can then be used to merge ontologies, migrate data between them or translate requests made in one ontology to another.

A new technique for ontology alignment has been developed, which integrates some important features in matching to achieve high quality results that will help in searching and exchanging information between ontologies.

This is an ontology alignment system for solving the key issues related to heterogeneous ontologies, which will be used to resolve the heterogeneous

interoperability between ontologies in order to achieve better knowledge sharing and reuse. Moreover, it uses combination-matching strategies to execute the ontology-matching task. It can also be used to discover the matching for both ontologies. Moreover, our ontology alignment algorithm compares each pair of ontology terms, first by extracting their ontological contexts and second by combining different elementary ontology matchers. Indeed, the majority [26, 75] of such approaches use a combination of terminological and structural methods, where the lexical overlap is used to produce an initial matching which is subsequently improved by using the structure of source and target.

In our system, the match result is a set of alignment elements specifying the matching ontology elements together with a similarity value between 0 (different) and 1 (identical), indicating the reasonableness of their correspondence. This system focuses on one-to-one (1:1) and one-to-many (1: m) match relationships. On the other hand, match algorithms may determine multiple match candidates with different similarities for an ontology element, in order to select one of them. Indeed, a method have been generalised to admit any two ontologies and a threshold value as input. Comparisons among all pairs of ontology terms are established, producing as output an OWL, XML or HTML document with the alignments obtained. A filter is used to improve the alignment results.

This is a semi-automatic system which enables syntactical and semantic interoperability among ontologies. Our goal is to reach the highest number of accurate matches**.** In terms of implementation, our tool has been developed in Java and is based on several other already existing codes, which are described next.

Several tools and methods have already been developed to support the discovery of relationships between entities in different ontologies in a given domain. Some approaches or tools are based on single-strategy map matching, but others based on more than one strategy; these are called multi-strategy or hybrid matchers. It seems that a multi-strategy algorithm is actually better than a single-strategy one, because it deals with and solves more than one critical problem. It can be said that a system or tool which has more than one matching strategy is likely to be more conveniently

applicable in different domains. In fact, most systems or tools use multiple matching algorithms to cope with different types of mismatch between ontologies; selecting the matching algorithms depends on the application domain.

Our technique relies on a well-established measure [13, 28, 38, 80] for comparing the entities of two ontologies which are combined in a homogeneous way. The strategies that are defined in our system can be classified into four categories: terminological-based, linguistic-based, heuristic-based and structure-based strategies.

The process starts with two ontologies (RDF/OWL) as input; these need to be aligned with one another. All entities of the first ontology are compared with all entities of the second ontology. Before the comparison of entities (concepts, relations and instances) can be processed, it is necessary to choose an entity from each of the ontologies. In order to compare them, the two entities need to be extracted from extensional and intentional ontology definitions. In general, the obvious technique is to compare all entities of the first ontology with all entities of the second ontology. This process is based on most of the ontology features; in fact, a top-down strategy is adopted. In order to make an efficient mapping algorithm, several measures are used in the processing.

The system starts by loading two ontologies and extracts useful features such as class names, property names and subsumption relationships from them. There are then four matching steps.

- **Terminological Matching** is used to compute similarities based on the strings of class and property names. In fact, the most important feature for matching is the label.

- **Linguistic Strategies** combining numerous feature-matching approaches allow radically higher quality matching. Therefore, in this part the string matcher is used with the thesaurus in order to find synonyms and hypernyms.

- **Structural Matching** is a method used to compute the similarity between two classes by using graph information like compute the similarity between the super-classes of the two classes.

- **Heuristic-based Strategies** constitute a kind of matching where a string matcher is combined with a structure matcher in order to obtain high quality results.

Figure 4.1 illustrates the structure of our system.

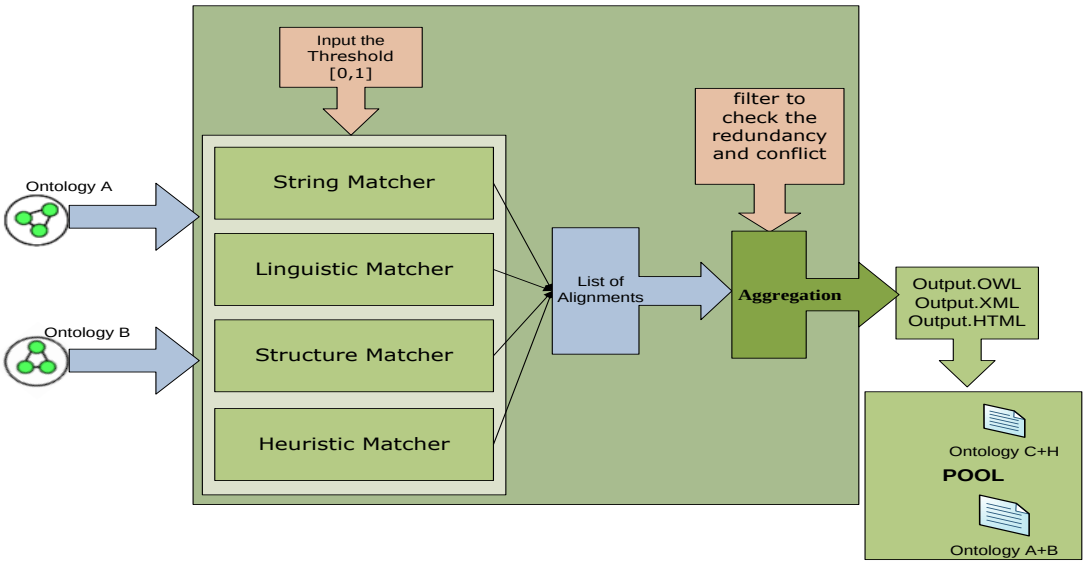


Figure 4.1: The main components of the system

4.2 Structure of Alignment Framework

This section introduces a computational framework for the alignment of ontologies and describes the main process of ontology alignment. So it outlines the most important features and elements required by an alignment system.

4.2.1 Pre-Processing

The main aim of this step is obtaining useful information from the ontologies that are to be matched, beginning by loading two ontologies and extracting useful ontological features such as class names, property, etc.

Input

The inputs for the alignment approach are the two ontologies O_1 and O_2, (e.g. OWL or RDF). The different elements of ontologies, e.g. concepts, instance and relations, can be aligned.

Common languages to represent ontologies are RDF(S) or OWL, although one should note that each language offers different modelling primitives. A higher expressivity of the ontologies, such as those provided by OWL constructs, would be favourable. More semantic information allows for more actions to identify alignments.

Feature Engineering

The main point of this step is obtaining useful information from the ontologies that are to be matched, beginning by loading two ontologies and extracting useful ontological features such as class names, property, etc. Normalisation is then carried out on these elements by removing stop words, for example.

The selected features are specific to an alignment generation algorithm. In general, the features of ontological entities (concepts *C*, relations *R* and instances *I*) need to be extracted from intentional and extensional ontology definitions. In fact, interpreting the ontologies is not as graphs only, but seek to exploit the semantics of each individual feature for our purposes. Labels are the most common feature used when considering related work approaches.

Search Step Selection

Before the comparison of entities can be initiated, it is necessary to choose which entity pairs (e, f) from the ontologies should actually be considered.

The most common methods for choosing candidate alignments are to compare:

- all entities of the first ontology O1 with all entities of the second ontology O2:

$$(e, f) \in E1 \times E2;$$

- or only those entities of the same type (concepts, relations, and instances):

$$(e, f) \in (C1 \times C2) \cup (R1 \times R2) \cup (I1 \times I2).$$

4.2.2 Alignment Process

The input of this step is useful features of the input ontologies. The purpose of this step is to find similarities through a group of different matching strategies. Finally, the output of this stage is a set of matching elements.

Similarity Computation

The main step is the computation of similarity for each pair of terms based on using a wide range of similarity functions to compute the similarity between an entity of O_1 and an entity of O_2. Similarities values actually represent evidence that how well the corresponding entities in O_1 match their counterparts in O_2, thus, can be aligned if they are identical. The similarity values range between 0 and 1. Each similarity function or similarity matcher is carried out differently and is composed of the introduced feature (concept, property or individual) existing in both ontologies and the respective similarity measure.

In fact, the most important step during matching discovery is the execution of multiple independent matching strategies based on different similarity measures. Each strategy calculates the similarity values between any candidate matching entities ($e1$; $e2$) based on their definitions in $O1$ and $O2$ respectively. Each strategy provides a matching result according to the similarity value between 0 and 1 for each possible candidate matching. Finally, the result of the matching execution phase with k strategies, x entities in $O1$ and y entities in $O2$ is a $k \times x \times y$ cube of similarity values,

all of which are stored in the repository for later strategy detection and combination steps.

Basically, ontology matching methods are defining a distance between entities and computing the best match between ontologies.

Formal definition of similarity must be given in order to align two ontologies; therefore, similarity between two entities x and y of an ontology (x and y can be concepts, relations or individuals) is defined as a distance measure.

A similarity measure is a real-valued function, sim (e_i, e_j): $O_1 \times O_2 \rightarrow [0, 1]$, measuring the degree of similarity between x and y.

4.2.3 Post-Processing

Again, the output of the previous stage is the input for this stage; in this case, sets of matching elements. The process is to find the aggregations between these sets and to filter them in order to remove any redundancy. Finally, the output is a set of alignments.

Similarity Aggregation

With several matching strategy algorithms, there are a number of similarity values for an applicant matching (*e*1; *e*2). For example, one is the similarity based on their labels (names) and another is based on similarities of taxonomic structure. For that reason, this step will extract the combined matching result from the individual strategy results stored in the similarity matrix. For each combination of ontology entities, the strategy-specific similarity values are aggregated into a combined similarity value [23].

Aggregating different similarities is pervasive in ontology matching systems which contain multiple individual matchers. Many strategies, e.g. [23] Max, Weighted and Average have been proposed to aggregate different similarities in the approaches.

- **Max [23]:** This strategy selects one from the maximum end of a range of similarities to be representative of the final similarity, which is too optimistic, especially in cases of contradicting similarities.

- **Weighted [23]:** This strategy determines a weighted sum of similarity values of the individual matchers and needs relative weights, which should correspond to the expected importance of the matchers. It overcomes the drawbacks of the *Average* strategy by assigning relative weights to individual matchers.

- **Average [23]:** This strategy returns the average similarity over all individual matchers, so it considers the individual similarities to be equally important and cannot distinguish between them. Indeed, it represents a special case of the weighted strategy.

Interpretation

The actual alignment is driven from the aggregated similarity values.

As a result of the previous step, this step uses the individual or combined similarity values to extract matching between entities from the source to the target ontology. In general, several mechanisms using thresholds or maximum values for similarity matching perform relaxation labelling, or other criteria. The matching process supports an optional designer interaction phase for mapping correction. The output is an alignment table which includes multiple entries of align ($e1$; $e2$) from $O1$ to $O2$.

Assigning the alignment is based on a threshold θ which is applied to the aggregated similarity measures. Each entity may participate in either one or multiple alignments. Every similarity value above the cut-off indicates an alignment, while every value below the cut-off is dismissed.

- **Thresholds**: Threshold-based filtering would allow us to retain only the most similar entity pairs.

- **Constant Similarity Value**: For this method, a fixed constant represents the threshold.

$$\theta = const$$

The constant threshold seems reasonable, which is considered as evidence for alignments. If too little evidence is extracted from the ontologies, it is simply not possible to present reliable alignments. However, it is difficult to determine this value. One possibility is an average that maximizes the quality in several test runs.

- Many of the techniques used are based on computing a distance or a similarity between ontology elements. For common similarity measures, similarity ranges from 0 to 1. A score of 0 means the items compared are totally different, while 1 means that they are identical. Our design follows this rule by normalising each semantic distance that is smaller than the threshold as:

$$Semantic\ Distance(C, C') \leq Threshold$$

Output (Matching Representation)

The output of the process is a list of alignments. Given two ontologies, an output of alignments is created through:

Output: O1 x O2→E1 x E2 x [0..1] x {alignment}

The format and structure of the result of the alignment are specified for each method, whether it is the alignment that takes place between the entire structures or for couple entities of the two ontologies. The result for the majority of existing methods is an alignment file (usually in XML format), indicating couples, which are ontological entities that match. All methods of alignment determine connections between the entities using ontological measures of similarity.

In general, matching cardinality, which is used to view the overall match result, may explain that one or more elements from ontology A are related to one or more

elements in ontology B with different similarities, yielding four cases: 1:1, 1:n, n:1, n:m [28].

As mentioned previously, the matching process is divided into four basic steps returning semantic relations (≡,⊑,⊒,⊥,Idk) with similarity coefficients [0..1], which are often considered as equivalence relations with a certain level of plausibility or confidence.

In various approaches [26, 75], the aligned entities may be classes, properties or individuals. The discovered matching is encoded using standardised mapping representation languages, which provide the formalism for describing matching elements [28]. The alignment element is a *tuple* general definition, as follows:

5-tuple: (id, e1, e2, n, R),

where *id* is a unique identifier of the given matching element, *e1* (entity1) is the first aligned entity from the first ontology (concepts, properties, relations), *e2* (entity2) is the second aligned entity with the same constraint as entity1 from the second ontology and *R* is the relation holding between the two entities (e.g. subsumption, equivalence relation). The relation *R* is defined in terms of the confidence measure.

- If $Sim(e1, e2) = 1$ then *R* is the equivalence (=) relation.
- If $Sim(e1, e2) = 0$ then *R* is the disjointness ($\perp$) relation.
- If $Sim(e1, e2) > t$ (threshold) then *R* is a subsumption ($\supset$ *or* $\subset$) relation.

The symbol *n* (strength) denotes the confidence held in this correspondence; it is a similarity measure in some mathematical structures (typically in the [0, 1] range) holding for the correspondence between the entities *e*1 and *e*2. Most often this reflects the confidence of the alignment provided in the relation holding between the entities.

4.3 Detailed Description of Alignment Framework

Before the ontology alignment process can start, the user needs to select two ontologies, O_1 and O_2, in the ontology management module, then choose the matching technique; next, the threshold amount with the output format is determined, then the command to start matching should be given.

The inputs are ontologies expressed in OWL or RDF and the output is a document in a particular format, such as OWL, XML or HTML. The first step is to extract the ontological context of each term involved.

4.3.1 Pre-Processing

Execution begins with the importing of ontologies from libraries or existing ontologies. Thus, this starts with the two ontologies that are to be aligned, from which the name of classes and relations are obtained and separated into lists, as shown in Figure 4.2.

The features of ontological entities (concepts *C*, relations *R* and instances *I*) are extracted from extensional and intentional ontology definitions in order to compare entities from two different ontologies. Each of the features should be used to calculate the similarity between ontologies, therefore, ontologies cannot be interpreted as graphs only, but seek to exploit the semantics of each individual feature for our purposes. Labels are the most common feature used when considering related work approaches.

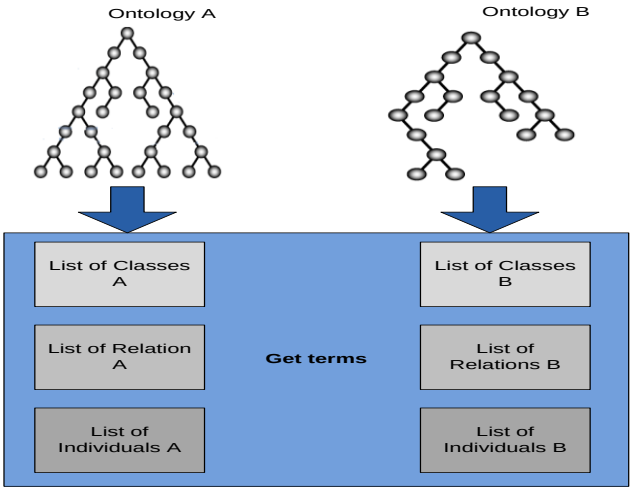


Figure 4.2: Getting the items from ontologies

In general, data element nametags in a heterogeneous environment such as the Web can be a single letter word (e.g. D for date), a combination of lexemes (e.g. FirstName, Dep_Number), a preposition (e.g. from, to) or a verb, which makes them syntactically diverse. Therefore, pre-processing of element names is necessary to improve the matching between ontologies; before the matching process starts, applying some translation process to each entity is required, in order to translate labels to concepts and to remove ambiguity. The first step is to convert each term to lower case to allow exact comparison of string names. The pre-processing of data element name tags comprises the following steps:

- Tokenisation of the entities: tokenisation consists of segmenting strings into sequences of tokens by a tokeniser, which recognises punctuation, cases, blank characters, digits, etc. Thus, names are parsed into tokens by recognising for example (Car-Driver name)⇒(car, driver, name).

- Lemmatisation: This finds all permutations of a word; e.g. Cars is associated with its singular form, Car.

- Elimination from multi-word terms of prepositions, conjunctions and resulting stop words, such as “a”, “the”, “of” and “in”. Tokens and labels are analysed in order to find all their basic forms, so those tokens which are not letters or

digits will be eliminated. Thus, in Dep_Number →{Dep, Number}, the tokens form a token set for each element.

4.3.2 Alignment Process

Ontology matching or alignment is carried out in order to specify matching relations among concepts from different ontologies. Such a rich set of semantic relations for expressing alignment is useful in ranking. This matching relies on similarity measures to establish alignment. In other word, ontology matching or alignment is the process of finding the closest semantic and intrinsic relationship between the existing ontologies of corresponding ontological entities. See figure 4.3.

The main step is the computation of similarity for each pair of terms based on using a wide range of similarity functions to compute the similarity between an entity of O_1 and an entity of O_2.

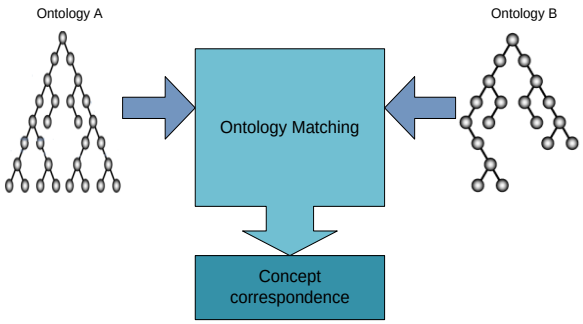


Figure 4.3: Ontology Matching

4.3.2.1 String-Matchers

Our prototype starts comparing lists by computing their literal semantic correspondence using a group of string-based criteria (i.e. edit distance, Jaccard similarity), so that only entities from the same category are compared for similarity. To do so, various entity categories are taken (classes, properties and instances) of each ontology and divided into separate lists; then the classes from the first ontology

are compared with classes from the second: properties vs. properties and instances vs. instances. If the similarity values of the comparison are greater than a predefined threshold, then inserting an element in the matrix with their degree of similarity is essential.

In the matching process it is important to use all available knowledge about the features of ontology entities. To this end, this system uses a group of well-known string matchers for each ontology feature. String similarity algorithms take as an input two strings and return a value indicating the distance or similarity between them. The more obvious method for relating two entities is to identify the label describing them.

This system uses an edit-distance-based strategy, which calculates the edit distance between the labels of two entities. Indeed, many measures, including edit distance and Jaccard similarity, have been proposed to cope with strings.

A similarity calculation is a type of matcher which plays an important role in the matching process. It calculates similarities between ontology entities through multiple string distance metrics. Similarity is computed between each entity from the source ontology and all entities from the target ontology. This matching type is based on the calculation of edit distance [44] between the names of two entities and uses the Jaro/ Levenshtein distance to capture the string similarity. String similarity between two entities commonly relies on their names, labels, comments and some other descriptions.

In this matcher the string-based matcher is implemented for identifying similar elements; it takes two strings as input and returns the similarity of these two with similarity values greater than a predefined threshold. They are used to identify identical classes of two ontologies based on the similarity of their names or descriptions.

Five strings-based matching are implemented in this system:

4.3.2.1.1 Levenshtein Edit Distance

The Levenshtein edit distance is a measure used to calculate the minimum cost of transforming one string into another by using editing operations: insertion, deletion or substitution. It is applied to these ontologies to discover correlations and to ensure accuracy and to reduce mismatches of element names.

Using the Levenshtein distance [58], similarity is defined as:

$$sim(s1, s2) = 1 - \left[\frac{edit\_distance(s1, s2)}{max\{length(s1), length(s2)\}}\right]$$

where edit-distance (s1, s2) denotes the string edit distance function between the two strings s1 and s2.

The edit distance is a dissimilarity δ: S × S → [0, 1], where δ(s1, s2) is the cost of the least costly sequence of operations which transforms s into t.

4.3.2.1.2 Jaro

Jaro is a string similarity metric used with a specific metric to calculate rapidly the distance between two terms. It is based on the number of common characters in two strings. The Jaro metric is provided by SecondString [97].

The Jaro measure is a non-symmetric measure σ: S × S → [0, 1] such that

$$\text{Jaro}(s1, s2) = \frac{1}{3}\left(\frac{|s1'|}{|s1|} + \frac{|s2'|}{|s2|} + \frac{|s1'| + \text{Tp}_{s1',s2'}}{2|s1'|}\right)$$

Where s1, s2: input strings; |s1'|: number of characters in s1 that are common with s2; |s2'|: number of characters in s2 that are common with s1; $\text{Tp}_{s1',\ s2'}$: number of transpositions of characters in s1' relative to s2'.

4.3.2.1.3 Jaccard Similarity

Jaccard Similarity [51, 97] is based on instance matching. The easiest way to compare classes when they share instances is to check the intersection of their instance sets S and T. The results will depend on the set relations; for example, these classes will be very similar if S $\cap$ *T* = S = T, more general when S$\cap$ *T* = T or S and disjoint when (S $\cap$ *T* = $\emptyset$).

$$JaccardSim(S,T) = \frac{P(S \cap T)}{P(S \cup T)}$$

or

$$JaccardSim(S,T) = \frac{P(S,T)}{P(S,T) + P(S,\neg T) + P(\neg S,T)}$$

This measure is normalised and reaches 0 when S ∩ T = ∅ and 1 when S = T. It can also be used with two classes of different ontologies sharing the same set of instances.

4.3.2.1.4 SoftTF/IDF

SoftTF/IDF applies to ontology-external features, a class which subsumes any kind of information not directly encoded in the ontology, such as a bag-of-words from a document describing an instance. In general, a class has a long descriptive text comprising comments or descriptions, which provides a human-readable text that describes what that class is. So, this method used to compute the relevance between classes based on the comments. Actually, this does not measure similarity, but assesses the relevance of a term to a document. Also, it is a well-known method used in many domains such as information retrieval and classification of domains.

TF/IDF [97] is defined as:

$$V'(w,S1) = \log\left(TF_{w,S1} + 1\right) * \log(IDF_w)$$

Where $TF_{w,\ S1}$ is the frequency of word w in S1; IDF_w is the inverse of the fraction names in the corpus that contain w.

A string similarity metric called *soft TFIDF* is used as implemented in the SecondString open-source software. TFIDF-based distance metric, extended to use "soft" token-matching.

$$SoftTFIDF(S1,T) = \sum_{w \in CLOSE(\theta,S1,T)} V(\omega,S1) * V(\omega,T) * D(\omega,T)$$

Let CLOSE (θ, S1, T) be the set of words w∈S such there is some v∈T such that dist'(w, v) > θ and for w∈CLOSE(θ,S1,T), let $D(w,T) = \max_{v \in T} dist(w,v)$

$$V(\omega,S1) = \frac{V'(\omega,S1)}{\sqrt{\sum_{\omega'} V'(\omega,S1)^2}}$$

4.3.2.1.5 Soundex

Soundex techniques establish equality between the names of elements based on how they sound. For example, the elements "to let" and "2 let" are different in spelling, but similar in sound. This matcher invokes Soundex distance methods provided by similarity metrics, computing the phonetic similarity between names from their corresponding Soundex codes [97]. Soundex is produced by the Web Intelligence Group at the University of Sheffield.

4.3.2.2 Linguistic Matchers

Natural language processing is a technique that uses the morphological properties of words to identify important concepts within a source and to compute the linguistic meaning of the label. These steps are supported by an external dictionary (such as WordNet) to verify whether two concepts are equal or similar, as shown in Figure 4.4. Linguistic resources are used to reduce the mismatching that could arise from the existence of synonyms or hyponyms. Here, WordNet is used to provide a source of synonyms, hypernyms and hyponyms.

First, linguistic-similarity matches are applied for the initial comparison, using the measure of linguistic similarity among concept names to solve term matching. Therefore, mapping the source ontology of class names, values of attributes and

relation names to target ontology class names, values of attributes and relation names is needed.

Linguistic similarities are computed by examining the similarities between the local descriptions of the classes. As a basic group of matching techniques, they are usually the initial step to suggest a set of raw mappings with which other matchers can work. These use additional linguistic resources such as lexicons and thesauri in order to identify synonyms.

The semantic measure determines the meaning of the terms, which includes information such as synonyms and hyponyms. In this phase, an attempt is made to find a common element in the synsets of two names.

This phase also generates a synset for each element that includes the synonym set retrieved from WordNet. Two terms may be similar, even if they are spelt differently. The retrieval of the synonym set (e.g. car → automobile) is an example of the use of synonyms. In general, the names of two nodes having a related sense are expected to be somehow related.

The lexical semantics similarity measure explores the semantic meanings of the word constituents by using external resources, such as user-defined lexica and/or dictionaries to help identify synonyms in matching.

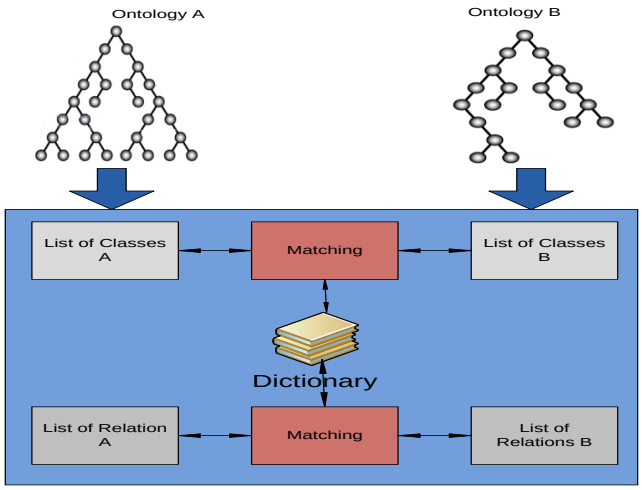


Figure 4.4: Lexical-based matching

Semantic interoperability is about ensuring that the precise meaning of the information exchanged can be understood by other systems, especially if they are not tailored for this specific information exchange.

In an OWL ontology, properties such as “sameIndividualAs” or “sameClassAs” could be used to show that those two entities are the same.

Figure 4.5 shows the two types of matching used in our prototype.

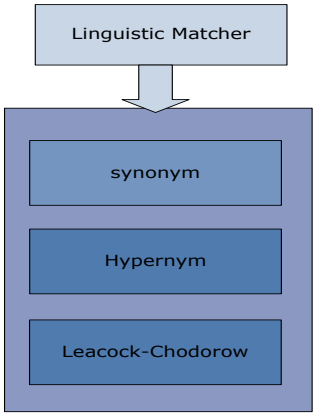


Figure 4.5: Prototype linguistic matcher

In general, ontological semantic heterogeneity appears in ontologies as concepts for a domain; these are described with different terminologies (synonymy). For instance, 1) the terms *booking* and *reservation* are synonymous but termed differently; 2) different meanings are assigned to the same word in different contexts. Homonyms are identical words used to name different entities. On the other hand, structural heterogeneity among ontologies appears from different taxonomic structures.

4.3.2.2.1 Synonymy

Synonyms are different words used to name the same entity. For instance, “car” and “automobile” are synonyms in some contexts. In order to help solving the problem of using different terms for the same concept in the ontologies, thesaurus must be used.

A simple measure of synonymy as similarity is as follows. Given two terms s and t and a synonym resource Σ, the synonymy is a similarity σ: $S \times S \rightarrow [0, 1]$ such that it

takes as input two strings, splits them into tokens, removes stop words, retrieves synonyms for each of them and compares the two sets of synonyms. The similarity is computed by:

$$\frac{2 \times \sum_{i\ th\ token\ of\ C1, j\ th\ token\ of\ C2} \mathrm{Jaro}(i,j)}{|C1| + |C2|}$$

where *C*1 is the first Class name, *C*2 is the second Class name, Jaro returns the string distance between the *i*th and *j*th token from the respective names, and |C1| and |C2| are the lengths of C1 and C2 respectively.

$$\sigma(s,t) = \begin{cases} 1 & \text{if } \sum(s) \cap \sum(t) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

4.3.2.2.2 Hypernym

The hierarchy uses a thesaurus such as WordNet [32]; it builds a path between two terms and calculates their similarity considering this path through hypernym relations. In fact, a combination of lexical matching with the WordNet dictionary will offer more accurate semantic similarity results. Many efforts have been made in this area, such as that of Leacock-Chodorow for WordNet; therefore it is used here to add efficiency and obtain high quality results. The Leacock-Chodorow measure calculates a taxonomic path length between two words.

Our system uses WordNet, which is a free electronic lexical database for some natural languages; it lists semantic and lexical relations between words, where various senses of words are grouped together into sets of synonyms called synsets. Many synsets are connected to each other via a number of semantic relations. As a result, we obtain concepts that have the source ontology term as their synonym. Indeed, a similarity value of 1 is assigned if the source ontology terms are synonyms of the same concept, and 0 otherwise. While the matcher is based on terms, it uses a general thesaurus, WordNet, to enhance the similarity measure by looking up the hypernym relationships of the pairs of words.

4.3.2.2.3 The Leacock-Chodorow Matcher

The Leacock-Chodorow matcher exploits the Leacock-Chodorow semantic similarity measure [53], which measure assumes a virtual top node dominating all nodes and will therefore always return a value greater than zero, as long as the two concepts compared can be found in WordNet, since there will always be a path between them. It returns "=" if the measure exceeds the given threshold and "Idk" (I do not know) otherwise. No corpus data are used by this measure, so it cannot be affected by sparse data problems.

The measure is based on counting the number of links between two input synsets. Intuitively, the shorter the path, the more related are the concepts under consideration. Leacock and Chodorow considered nouns in a hierarchy and proposed the following formula for estimating the similarity of two synsets:

$$sim_{lc}(c1, c2) = -\ln\left(\frac{C.PathTaxo(c1, c2)}{2 * D}\right)$$

where C.PathTaxo(c1, c2) is the length of the shortest path between the two synsets c1 and c2, and D is the depth of the tree. The measure has a lower bound of 0 and upper bound defined as follows:

$$Up_b = -\ln\left(\frac{1}{2 * MaxDepthTaxo}\right)$$

where MaxDepthTaxo is the maximum depth of the taxonomy.

4.3.2.3 Structure Matching

In fact, a lexical similarity measure is not sufficient; therefore, many rules of ontology structure are combined, using the structure of concepts and properties to help in finding similarities between ontologies.

The matching of ontologies based on their relational structure is considered to be very powerful, because it allows all the relations between entities to be taken into account.

Thus, it must be grounded in other real properties; moreover, it is usually used in combination with (internal) structural methods and terminological methods, so if the properties of two concepts are equal, the concepts are also equal. If the domains and ranges of two properties are equal, the properties are also equal. A graph-based module that uses domains and a range of properties is built to find equality between them. It is worth considering the important relations before using any techniques. As mentioned previously, taxonomy, which has attracted much attention from researchers, is the most commonly used structure, because it is the backbone of ontologies. Taxonomy-based modules consider only the specialisation (is-a) relation.

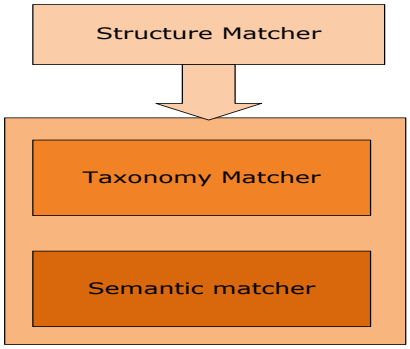


Figure 4.6: Prototype Structure Matcher

4.3.2.3.1 Taxonomy-Based Matching

Taxonomy-based matching considers the specialisation relationship only, in which "is-a" relations exist through nodes that are already similar, and in which neighbours are then also likely to be similar. When connecting a source of taxonomy to a super-element taxonomy target, the degree of generality of the link is supposed to be the same as in the "is-a" link between this super-element and other sub-elements in the taxonomy target. Thus, the taxonomy of asparagus in the source may be connected to that of fresh fruit and vegetables.

In general, class hierarchies are often represented in directed acyclic graph structures, and *is-a* relations are the significant built-in relations in class hierarchies and a number of *rdfs:subClassOf* relations. Indeed, based on these class hierarchies, linguistic similarities can be found among local descriptions, e.g. local names, labels and comments. Concept taxonomies are implemented differently in frame-based and DL-based languages. In frame-based languages, the *subclass-of* relationships between concepts must always be represented explicitly at the time of design, while in DL-based languages, the inference engine (usually called a classifier) can infer them at run time, even if they are not represented explicitly. The representation of *disjoint* and *exhaustive* knowledge in concept taxonomies will also be explained.

In order to match concepts between two taxonomies, a well-established measure to facilitate the evaluation of our system is applied. In addition, it allows us to influence special-purpose techniques for the matching process. In fact, our work in structure matching based on [8] with changing the lexical matching in order to improve the result. Lexical similarity is calculated by using Jaro-Winkler metric (JW) [15], which works mainly on the number of characters in common and also on the order characters between the inputs strings.

Maintaining a general rule that will help us in this kind of matching is needed, whereby two concepts are similar only if:

- Their sub-concepts are the same. Or
- Their super-concepts are the same.

The similarity between the sub-concepts or super-concepts of two concepts $e1$ and $e2$ is defined as follows:

$$Sim_{Supc/Subc}(e1, e2) = \frac{\sum_{i=1}^{n}(\sum_{j=1}^{m} sim(e1_{hi}, e2_{hj}))}{(n+m)/2}$$

where

- *hi* is the sub-concepts or super-concept i of $e1$

- *hj* is the sub-concepts or super concept j of *e2*
- *n* is the number of super-concepts or sub-concepts of *e*1
- *m* is the number of super-concepts or sub-concepts of *e*2.

Because it is a matrix, that required the use of the following formula in order to normalise the result:

$$Sim_{Taxo}(e1, e2) = W_{subc} \times Sim_{subc}(e1, e2) + W_{supc} \times Sim_{supc}(e1, e2)$$

where

W_{subc} and W_{supc} are the weights which indicate respectively the importance of the similarity methods.

Sim_{subc} and Sim_{supc} are that such $W_{subc} + W_{supc} = 1.00$.

4.3.2.3.2 Semantic Matching

This algorithm is based on a combination of methods which use the definition of the concept and its structure. The definition of the concept is the main consideration when mapping the concept of an ontology based on names, descriptions and relations; the conceptual structure method considers the concept of hierarchy among areas such as the relationship between nodes (parent node, sub-node) and semantic relations between neighbours.

The information which is already available at the nodes should be used: for example, attributes such as data type, range and domain. By using as much information as possible on the features of an ontology, the similarities between ontologies can be found based on the structure of concepts (their properties and relations) and of properties (domain, range and constraints).

In this kind of matching, a general rule which states that any two concepts are equal if their properties are equal needs to be maintained. In order to compute the similarity

between two concepts *e*1 and *e*2 based on their structures, the following formula is used:

$$Sim_{stru}(e,f) = \frac{\sum_{i=1}^{n}\left(\sum_{j=1}^{m} sim_{prop}(e_{pi}, f_{pj})\right)}{(n+m)/2}$$

where

- *pi* is property i of *e*1.
- *pj* is property j of *e*2.
- *n* is the number of properties of *e*1.
- *m* is the number of properties of *e*2.

In order to compute the similarity of two concepts based on the structure of their properties, two rules should be followed. First, two properties are equal only if:

- They have the same name, and /or
- The domains and the ranges of the two properties are equal.

Secondly, the similarity between two concepts e_{pi} and f_{pj} is given by:

$$Sim_{prop}(e_{pi}, f_{pj}) = \max\left(Sim_{lexi}(e_{pi}, f_{pj}), \frac{Sim(e_{pi\,dom}, f_{pj\,dom}) + Sim(e_{pi\,ran}, f_{pj\,ran})}{2}\right)$$

where

- pi^{dom}, pi^{ran} are the domain and the range of the properties *pi* and
- pj^{dom}, pj^{ran} are the domain and the range of the properties *pj*.

Lexical Similarity metrics is one of the methods of string matching (e.g. Jaro-Winkler [15]). In technical terms, matching two classes should begin by comparing

their names by using a string matcher (e.g. Jaro-Winkler); as a result of this step, the tool stores the label similarity as the initial similarity. Subsequently, all the related properties are retrieved, extract the domain and range of each related property, after that compute the similarity of the domain and range; this technique is useful for refining the initial similarity. In this process, we take into account the subsumption structure that is obtained from comparing domains and ranges of properties, and finally comparing the result with that of the previous step, which is label similarity (name matching).

Many groups of result can be obtained from this step (e.g. same property name, same range, but different domain). In order to deal with this, an arbitrary numeric value to each of these results is assigned; thus the same value to the same property name, domain and same range, and different values to different property names, domains and ranges are given.

For example, the classes Human and Person are equivalent, as the two attributes are very similar.

4.3.2.4 Heuristic Matching

The name of a class or property is considered one of the most important indications of its relevance; therefore, this step focuses on finding relations between terms belonging to different ontologies, based on their names. This technique begins by comparing class names, property names and instance by using an editing distance and substring distance between the entity names. Next, a distance matrix is built in order to choose the alignment from the distance; after that, the aggregates of these distances are applied with the symmetric difference of properties in classes. In fact, the structure of heuristic matching is similar to [30], with the addition of more features like super-class and relations, and the use of different lexical matching in order to improve the result.

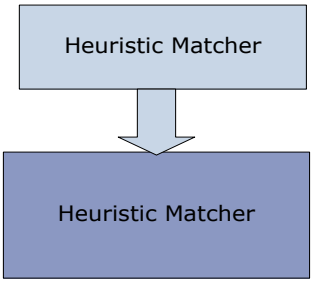


Figure 4.7: Prototype Heuristic Matcher

A heuristic for a relation "Sim_H" is a function

$$Sim_H\ (e, e'): O_1 \times O_2 \rightarrow [0,1]$$

$\forall\ e, e' \epsilon (E), Sim_H(e, e', O_1, O_2) \geq 0 \quad : (Positiveness)$

$Sim_H(e, e', O_1, O_2) = 1 \ : the\ relation\ H\ holds\ between\ the\ two\ entities$

$Sim_H(e, e', O_1, O_2) = 0 \ : the\ relation\ H\ \ does\ not\ hold$

$0 < Sim_H(e, e', O_1, O_2) < 1 \ : The\ relation\ H\ holds\ to\ a\ certain\ degree.$

Given two ontologies O_1 and O_2, each describes a set of entities, in an attempt to find corresponding entities (classes, properties, instances, etc.) e and e' with the same intended meanings in both ontologies

In fact, this matcher can work alone and provide a very good result, because it contains all the components of the system.

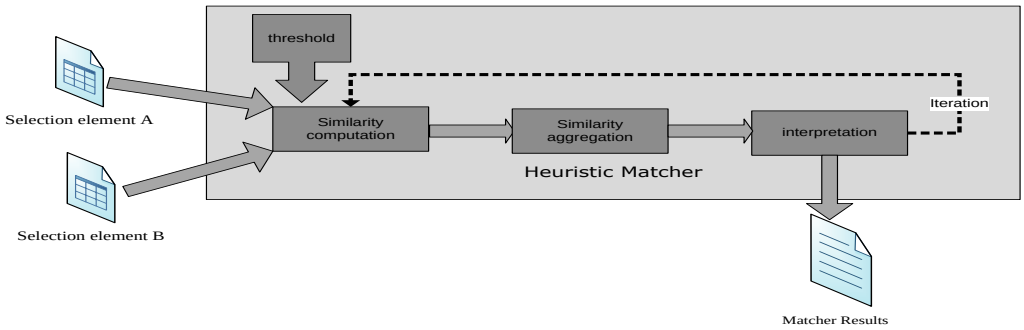


Figure 4.8: Heuristic Matcher

Figures 4.7 and 4.8 illustrate this matcher, whereby starting from two selection documents then execute the following equation:

$$Tot_{Sim(c,c')} = w * Sim_C(c,c') + w * Sim_P(c,c') + w * Sim_R(c,c') + w * Sim_I(c,c') + w * Sim_S(c,c')$$

where

- Sim_C (c, c') is the similarity between labels of classes,
- Sim_P (c, c') is the similarity between properties of classes,
- Sim_R (c, c') is the similarity between relations of classes,
- Sim_I (c, c') is the similarity between instances of classes,
- Sim_S (c, c') is the similarity between super-classes of classes,
- w is the weight and
- $Tot_{Sim(c,c')}$ is the average of all of these similarities, which is between [0,1].

Interpretation

Use all aggregated numbers as a threshold strategy to propose the equality for the selected entity pairs.

Iteration

Iteration is the core of our heuristic matching step. Thus, to calculate the similarity for one entity, should compute the similarity more than once from different perspectives. In order to compute the similarity between terms, in that situation, the first round uses comparison methods based on labels and string similarity.

By doing the comparison method over several rounds, one can access the pairs already computed and use more complex structural similarity measures. The iterations are done to find mappings based on lexical knowledge and then on knowledge structures. Our system iteratively executes this until no new matches are discovered.

From all the calculated similarities, an aggregated similarity value is obtained which expresses the confidence that the entities compared are the same; i.e. they can be aligned. A general threshold is set and all similarity values above the threshold automatically lead to an alignment, while all below lead to a non-alignment. In addition, the interpretation step is not critical with respect to efficiency. A threshold is determined and bijectivity of alignments is maintained.

In the process, the aggregation of single methods is performed once per candidate alignment, so there is no critical need to apply overall efficiency. Therefore, the sigmoid function is used following a function with manually assigned weights in this step. In our system, the weight is assigned automatically to 1.00.

The heuristic matcher returns both class and property mapping candidates as output. Hence, it is necessary to differentiate classes and properties by checking the type of entities returned using the semantic model.

4.3.3 Post-Processing

4.3.3.1 Aggregation

In general, in order to discover matching, multiple matching algorithms based on several similarity measures should be executed (such as names, structure or external information). The main task of these algorithms is to determine similarity values between candidate mappings (e1, e2) based on their definitions in source ontology and target ontology (O1, O2) respectively. Each matcher determines an intermediate matching or alignment result for each possible candidate matching with similarity value between [0..1]. The result of the matching execution phase with k matching algorithms, m entities in O1 and n entities in O2 is a k × m × n matrix of similarity values, which is stored in the repository for later combination and selection steps. The value of each matrix entry specifies the similarity of the specific pair to which the entry corresponds. The match result from the previous step can be aggregated into a single similarity value for two ontologies, called combined similarity. This depends on the chosen matchers and their combination strategy. To determine the best match candidate(s), the correspondence is ranked according to their similarity values per element and applies a filter strategy to determine the most reasonable ones. Figure 4.9 illustrates these steps.

As with any other tool, emphasis is placed on strong individual similarities by weighting individual similarity results, first with a sigmoid function and then summing the modified values to produce an aggregated similarity value. Next, a matrix M with all similarities is obtained, the different contributions are weighted and a final degree of similarity is provided.

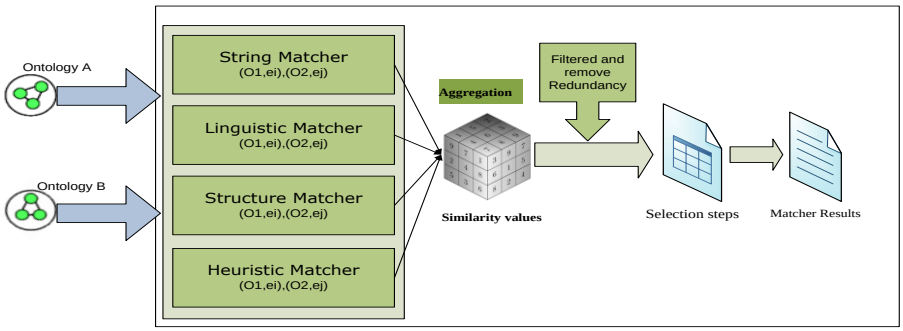


Figure 4.9: Aggregation Steps

4.3.3.2 Output (Set of Alignments)

The output of the alignment algorithm is a group of alignment relationships which holding between terms from the source ontologies. The similarity values are indicating how well the corresponding entities in O1 match their counterparts in O2.

This system is trying to find one-to-one and one -to -many mapping. On the other hand, some unmatched elements may occur, when:

- no correlation appears when there is no semantic overlap between the elements of source ontological and target ontological element (no mapping); and
- many-to-many mapping occurs when many elements of the source ontology has a correlation with more than one target ontological element.

The final alignment "A" is then extracted; finding the highest rated one-to-one relationships among terms and filtering out those that are below the given threshold. A *filter* selects the best mapping pairs from the resulting similarity matrix.

4.3.3.2.1 Filtering

All the previous steps provide a similarity set which contains a degree of similarity for each couple of resources; these results are also checked to avoid the redundancy

and mismatching that could appear; then they are weighted and used to perform the mapping task.

Filtering is a conflict checker which is used to avoid conflicts introduced by the alignment relationships. It is used to check whether any of the following conflicts occurs.

- Conflicts of concept names
- Redundancy of concepts.

This step involves user intervention, where the user invokes the following operation:

Given two ontologies O and O', alignments are prepared as a set of correspondences between pairs of entities (e, e') belonging to O and O' respectively.

A correspondence is described as a quadruple. Given two ontologies O and O', an alignment (A) between O and O' is a set of correspondences on O and O' with some additional metadata

4-tuple: (e, e', r, n) such that:

e, e' are the URI of some entity of the first ontology and the second ontology, respectively (e.g. XML elements, formulae, terms, classes, individuals).

r is the relation between e and e' asserted by the correspondence (e.g. equivalence (=), more general (<))

n is the degree of confidence in that correspondence (typically in the [0, 1] range) (value: string, see below), i.e. the relation holding between the first and second entities.

- $sim(x, y) \in [0..1]$
- $sim(x, y) = 1 \leftrightarrow x = y$
- $sim(x, y) = 0 \leftrightarrow x = \neg y$

4.4 Summary

A new technique for ontology alignment has been built by integrating some important features of matching to achieve high quality results when searching and exchanging information between ontologies. The system is semi-automatic and enables syntactical and semantic interoperability among ontologies. Our goal is to achieve the highest number of accurate matches. Our system is a multi-strategy algorithm which can deal with and solve more than one critical problem. Therefore, it is likely to be more conveniently applicable in different domains.

The process starts with two ontologies (RDF/OWL) as input; these need to be aligned with one another. Before the comparison of entities (concepts, relations and instances) can be processed, it is necessary to choose an entity from each ontology. The strategies that are defined in our system fall into four categories: terminological-, linguistic-, heuristic- and structure-based strategies. The match result is a group of alignment elements specifying the matching ontology elements together with a similarity value between [0, 1], indicating the reasonableness of their correspondence. In order to make an efficient mapping algorithm, several measures are used in the processing. Finally, the results are aggregated in order to determine the correct ones, which should be above the threshold.

In general, our prototype comprises three processes: pre-processing, alignment and post-processing.

Pre-processing (feature generation)

As a first step, useful information is obtained from the ontologies that are to be matched, beginning by loading two ontologies and extracting useful ontological features such as class names, property, etc. For example, normalisation is then carried out on these elements by removing stop words.

The inputs for this stage are two ontologies, in OWL or RDF. The process involves extracting important features to align them. Finally, the outputs are very useful features of these ontologies which are taken as input to the next stage.

Alignment (Group of Matchers)

In general, the similarity between entities needs to be calculated in order to find the correspondence between ontology entities. To do so, different strategies are described (e.g. string similarity, synonyms, structural similarity and similarity based on instances) for achieving similarity between entities.

The input of this step is useful features of the input ontologies. The purpose of this step is to find similarities through a group of different matching strategies. Finally, the output of this stage is a set of matching elements.

Post-Processing:

- Similarity Aggregator.
- Similarity Evaluator.

The output of the previous stage is the input for this stage; in this case, sets of matching elements. The process is to find the aggregations between these sets and to filter them in order to remove any redundancy. Finally, the output is a group of alignments.

Chapter 5

Implementation of the Alignment System

This chapter gives an account of how this system has been implemented in order to show how it works and what information it is able to deal with. There is also an analysis of the system components in order to show its efficiency and flexibility over different domains.

This part of the thesis elaborates how the system was developed. In fact, this was done independently, on the basis of several existing codes which were used in order to reduce the time needed for development, and because there was no need to develop aspects from scratch where there were existing codes which could be reused.

Thus, the following existing codes have been used in developing the system:

- The SecondString Project helped us in developing some of the string matching code (http://sourceforge.net/projects/secondstring/).

- SimMetrics is a similarity metric library which was also helpful in developing some of the string matching code (http://sourceforge.net/projects/simmetrics/).

- WordNet::Similarity (Java WordNet Library) helped in developing of the linguistic matching code (http://sourceforge.net/projects/jwordnet).

- The Alignment API is an application programming interface for expressing and sharing ontology alignments, which was used in developing the output format (http://www.inria.fr/index.en.html).

- The CROSI Mapping System, developed by the University of Southampton and HP Laboratories, provided some general ideas and structures in the matching, extracting useful information, and developing the output format. (http://www.aktors.org/crosi/deliverables/summary/cms.html).

5.1 System Structure

As noted in Chapter 4, our system is multi-strategies ontology alignment framework, because any tool which uses just one matching strategy is unlikely to achieve as many good match candidates as one that combines several strategies. Usually, combining several strategies can be done in two ways:

- Hybrid matching, which integrates multiple matching criteria. This approach determines match candidates based on multiple criteria or information sources, providing better performance and better match candidates than the separate execution of multiple matchers.

- Composite matching, which combines the results of independently executed matchers; in this approach it is possible to evaluate them simultaneously or in a specific order.

Figure 5.1 depicts the interface of our system.

5.1.1 Pre-Process

These components are then applied to the pre-process stage, illustrated in Figure 5.3, which begins by extracting the features of ontological entities (concepts *C*, relations *R* and instances I) from each ontology.

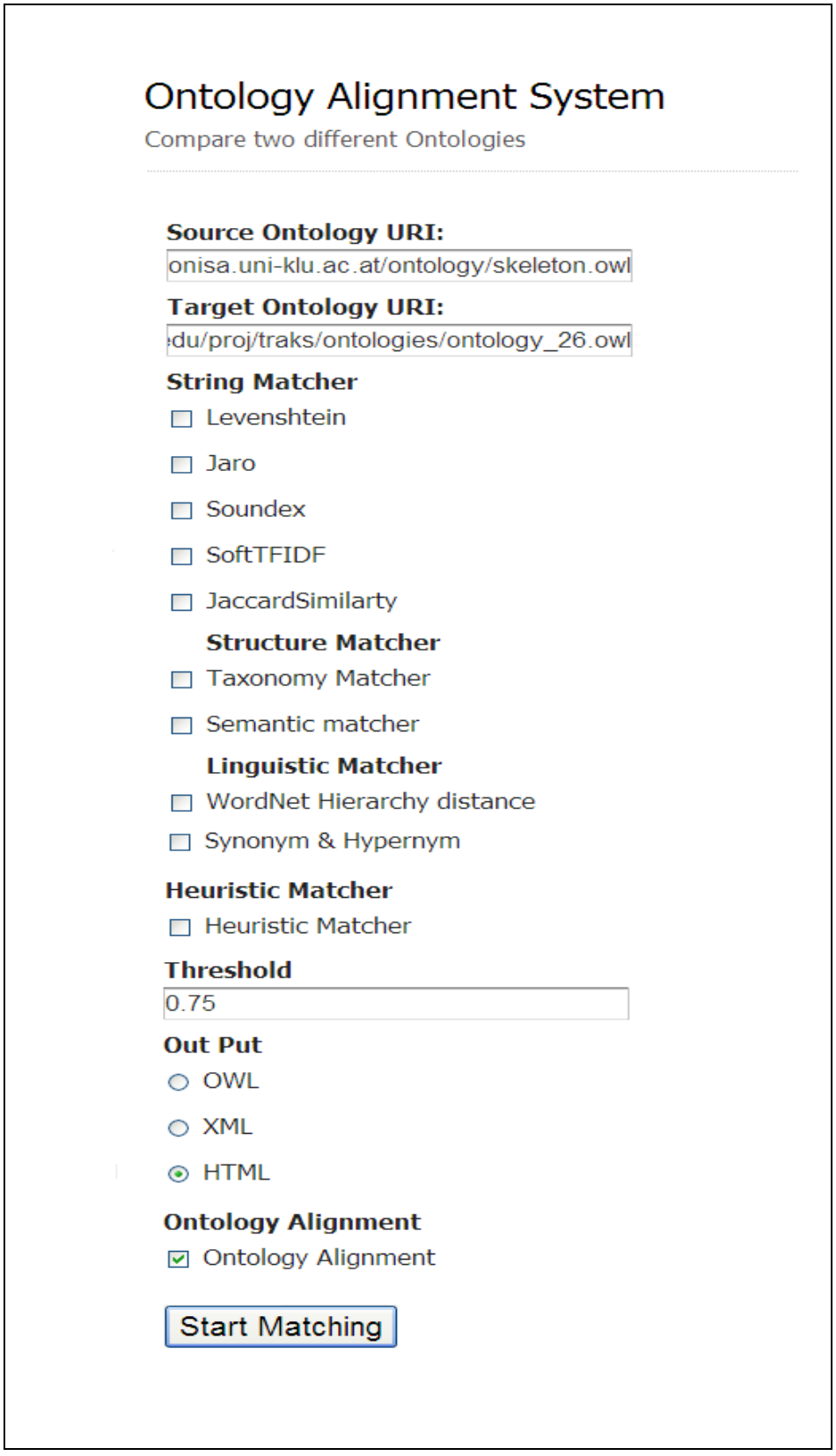


Figure 5.1: System interface

Figure 5.1 shows interface of our system which contains the following parameters:

- P1: URI of the source ontology (O1)
- P2: URI of the target ontology (O2)
- P3: a list of matchers
- P4: threshold
- P5: output formats

Therefore, the user should select these parameters as input:

- The two ontologies which are to be aligned; in other words, the inputs for this stage are two ontologies (e.g. OWL and RDF).

- The selection matchers, which could be one or more. In order to use our system should use the matcher in the bottom "Ontology Alignment" which performs all matcher together.

- The selection threshold, which should be between [0, 1]. This method consists in selecting correspondences over an exact threshold, applying a filter which retains only the most similar entity pairs.

- The output format (e.g. OWL, XML or HTML).

```
OntologyAlignmentSystem.run(new String[]{
"select source// http://www.????.owl/rdf"
"select target// http://www.????.owl/rdf"
"select matchers"
"select threshold between [0..1]"
"select output format [1,2,3]"})
```

Figure 5.2: The pseudo-code for run the system

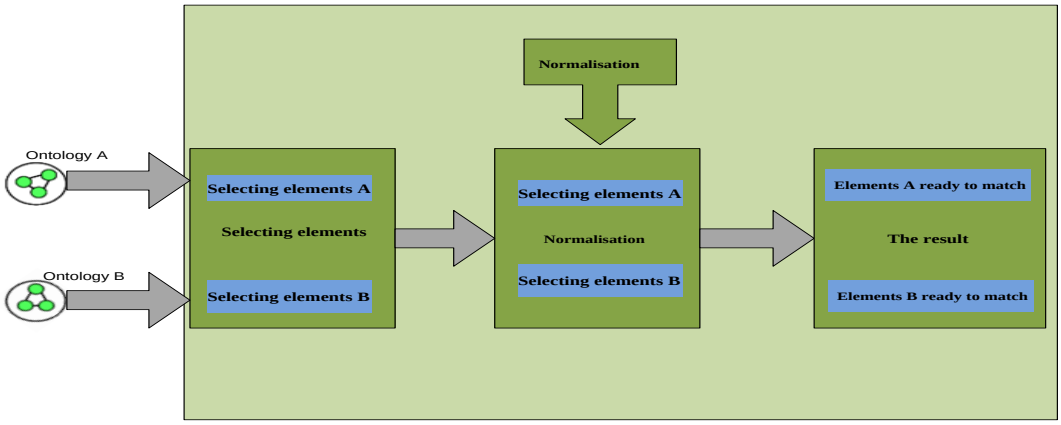


Figure 5.3: Pre-process stage

This system uses standard languages, such as OWL or RDF as input, which provide vocabularies to define the formal semantics of ontology. Thus, they use *owl:Class* and *rdfs:subClassOf* in order to define concepts and sub-concepts, and *rdfs:Property* and *rdfs:subPropertyOf in order* to define properties and sub-properties. They also use *rdfs:domain* and *rdfs:range* of a property to define what concepts can have the property and what instances of the concepts can be the values of the property. All these expressions help to extract element easily from ontologies.

From two ontologies or a list of features, ontology languages usually deal with the following kinds of entity:

1. Classes or concepts are the main entities of an ontology.

   1.1 These are interpreted as a set of individuals in the domain.

   1.2 They are introduced in OWL by the *owl:Class* construct.

2. Relations are the ideal notion of a relation independently of what it applies to.

   2.1 Relations are interpreted as a subset of the product of the domain.

2.2 These are introduced in OWL by the *owl:ObjectProperty* or *owl:DatatypeProperty* constructs.

2.3 Datatypes are exacting parts of the domain which specify values, as opposed to individuals; however, values do not have identities.

2.4 Data values are simple values.

3. Individuals, objects or instances are interpreted as exacting individuals of a domain.

3.1 These are introduced in OWL by the *owl:Thing* construct.

Figure 5.4: Class of Extracting Useful Features

Figure 5.4 illustrates the class that use to extract the features of each ontology.

The second action at this stage is to apply the normalisation process, examples of which are the following:

- **Tokenisation** consists of segmenting strings into sequences of tokens by a tokeniser, which recognises punctuation, cases, blank characters, digits, etc. Thus, names are parsed into tokens by recognising for example (Car-Driver name)⇒(car, driver, name).

- **Lemmatisation:** The strings underlying tokens are morphologically analysed in order to reduce them to find all their possible basic forms. Morphological analysis makes it possible to find flexion and derivations of a root. For example (Cars) ⇒(Car).

- **Stopword Elimination:** Tokens such as articles, prepositions and conjunctions (e.g. a, the, by, type of) are marked to be discarded, because they are considered non-meaningful for matching.

- **Normalisation** procedures are applied before comparing actual strings which have a meaning in natural language. These procedures can help to improve the results of subsequent comparisons. In particular, case normalisation is used to convert each alphabetic character in the strings into lower case; for example, “BOOK” becomes “book”.

- **Link Stripping** consists of normalising some links between words, such as replacing apostrophes and blank underline with dashes or blanks. For example, “per- introduction” becomes “per introduction”.

- **Punctuation Elimination** suppresses punctuation signs; for example, “C.D.” becomes “CD”.

In normalisation operations, certain factors should be taken into account; for example:

- Normalisation may reduce variation but increase synonyms, so some meaningful information may be lost; for example, “C.D.”, which is an

abbreviation for a particular organisation, becomes “cd”, which could have many other meanings.

5.1.2 Matching Process

The next stage is computation of similarity, considered the main stage in the alignment algorithm process, which computes the similarity between entities of O_1 and O_2 using a wide range of similarity functions. Thus, similarities represent evidence that two entities are the same and can be aligned. An alignment is therefore a one-to-one or one-to-many equality relation. Since both try to exploit lexical and structural information to find correspondence, ontology alignment often goes further, owing to the characteristics of ontologies.

Our technique relies on a well-proven measure for comparing the entities of two ontologies which are combined in a homogeneous way. The strategies that are defined in our system are classified into four categories: string-based, linguistic-based, heuristic-based and structure-based. Figure 5.5 depicts these main components.

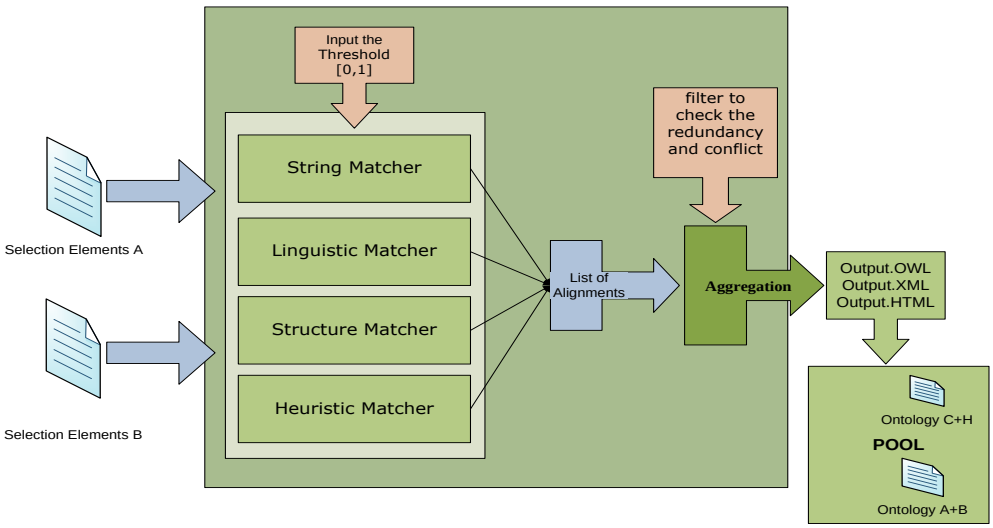


Figure 5.5: System Architecture

In the matching process, labels or identifiers are considered to be important in helping to align most of the entities; in general, this process is executed by string matching or language matching. Moreover, the structure can help to identify alignments. If an ontology is more expressive, then better matching results will be obtained.

5.1.2.1 String Matching

In general, the name of a class (i.e. label) is presented as a chain of characters without space characters. It is used to provide a human-readable description of class. Therefore, a name of class may be a word, or a combination of words. In fact, the name of each class should be unique in the ontology.

Terminological methods can be applied to the class names, the URIs, the label or the comments concerning entities to discover those which are similar.

In fact, string distance is often used to find correspondences between ontologies or to match names and name descriptions of ontology entities. If two ontologies share the use of at least one entity, then they may be compared. A string matcher usually takes as input the names of two concepts, then calculates the distance between them by distance functions that map a pair of strings to a real number. Consequently, the output will be a numeric value $c \in [0, 1]$ to represent the confidence of the similarity. The main reason for using such measures is the fact that similar entities have similar names and descriptions across different ontologies.

String similarities are based on the assumption that the names of concepts and properties representing semantic similarity will have similar syntactic features. Thus, the similarity approach analyses elements by looking at their data types (string, integer, float, etc). Identical class attributes have to be of the same data type. Data are classified by interpretation and analysis of key properties. Since instances are often included, it is possible to identify similar classes by looking at their instances. If two classes have the same instances, then they are expected to match.

Therefore, five well-known measures are used in order to calculate the degree of similarity between any pair of names over ontologies:

- Edit-distance (e.g. Levenshtein distance, Soundex , Jaro)
- Token-based Distance (e.g., Jaccard similarity)
- Hybrid Distance (e.g. SoftTF/IDF, Jaro-Winkler,)

These techniques are usually applied to names, labels, comments concerning entities and the URI. The scaled range is [0, 1] for comparing strings. To achieve high quality results and based on many experiments, the system disregards similarities that are smaller than a threshold of 0.65, and matches similarities greater than 0.65 to the full range [0, 1]. Figure 5.6 illustrates the classes of string matcher.

One of the important keys in ontology matching is how to choose the threshold. The answer is motivated from the idea that similarity is dependent on the domain of search or match. Therefore, if the domain is very important, like a medical domain, then the threshold should be high, that will lead to a reduction the group of matching and an attempt to get the exact right match, but if the domain is, for example a small business, then the threshold could be low; as mentioned before the 0.65 is the recommended value for reaching a very good result. Thus choosing the value of the threshold is very important, and therefore, it could be left to the expert to determine the value. So if the similarities value between terms exceeds the value of the threshold, then they are measured as similar.

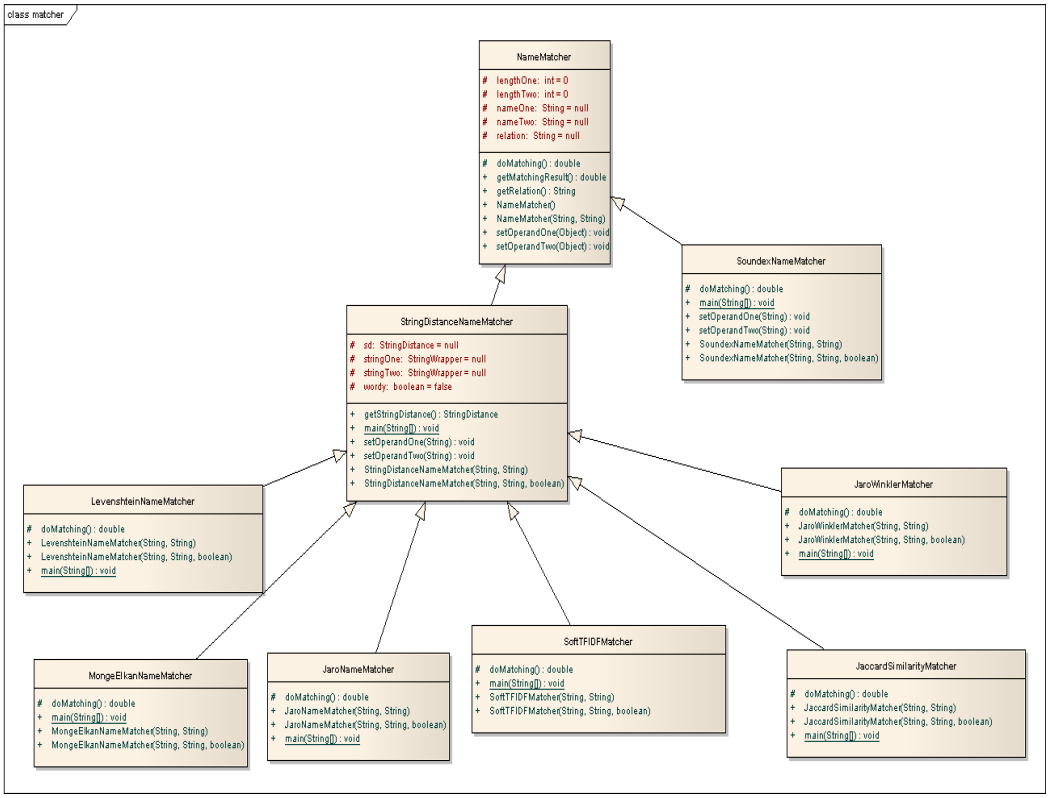


Figure 5.6: String matcher classes

Levenshtein:

The Levenshtein edit distance is a measure used to calculate the minimum cost of transforming one string into another by using editing operations: insertion, deletion or substitution in other words, the minimum number of edits required to transform one word into another. Therefore, it is applied to URIs, labels, comments and text from classes, properties and instance values as text sources.

In fact, the Levenshtein algorithm is very simple to implement. For example:

X= First string ("Car"), Y= second string ("Car"), then LD(X, Y) = 0. This means that the strings are already identical, because no transformations are needed.

X= first string ("Car"),Y= second string ("Bar"), then LD(X,Y) = 1; one substitution is sufficient to transform X into Y (change "C" to "B").

$$sim(t1, t2) = 1 - \left[\frac{edit\_distance(t1, t2)}{max\{length(t1), length(t2)\}}\right]$$

where edit-distance (t1, t2) denotes the string edit distance function between the two strings t1 and t2.

Algorithm 1: Levenshtein Algorithm.

```
1:/* fixed k to be the length of tI; */
2: /* fixed I to be the length of t2;*/
3:        if (k = 0) then
4:              return I;
5:         end if
6:       if (I = 0) then
7:             return k;
8:        end if
9:   /* build a matrix with I rows and k columns;*/
10:        for all letters of tI do
11:            for all letters of t2 do
12:               if (tI[a] = t2[a]) then
13:                    the cost is 0;
14:               end if
15:               if (tI[a] ≠ t2[a]) then
16:                    the cost is 1;
17:               end if
18:            d[a, b] = min (d[a − 1, b] + 1, d[a, b − 1] + 1, d[a − 1, b − 1] + cost);
/*the terms of the min signify correspondingly the insertion, the deletion and substitution.*/
19:               end for
20:          end for
21:  return d[k, l]; Sim (tI, t2) = 1 −distanceMax(|tI|, |t2|).
```

Figure 5.7: Levenshtein Algorithm

Jaro:

Jaro provides a rapid function to establish the similarity between two terms. In other words, this comparison is based on the number and order of the characters common to two strings. The following steps describe the Jaro algorithm:

- Compute the string lengths,
- Find the number of common characters in the two strings, and
- Find the number of transpositions.

$$\text{Jaro}(s1, s2) = \frac{1}{3}\left(\frac{|s1'|}{|s1|} + \frac{|s2'|}{|s2|} + \frac{|s1'| + \text{Tp}_{s1',s2'}}{2|s1'|}\right)$$

Where s1, s2: input strings; |s1'|: number of characters in s1 that are common with s2; |s2'|: number of characters in s2 that are common with s1; $\text{Tp}_{s1',\ s2'}$: number of transpositions of characters in s1' relative to s2'.

Example:

Jaro (JONES, JOHNSON) = 1/3 * (4/5 + 4/7 + (4 - 0)/4) = 0.790

Soft Token Frequency /Inverse Database Frequency:

A well-known procedure called a SoftTF/IDF is used in order to achieve matching based on the comments or descriptions which are used to describe the class or property. SoftTF/IDF calculates the similarity between these descriptions. See figure 5.8.

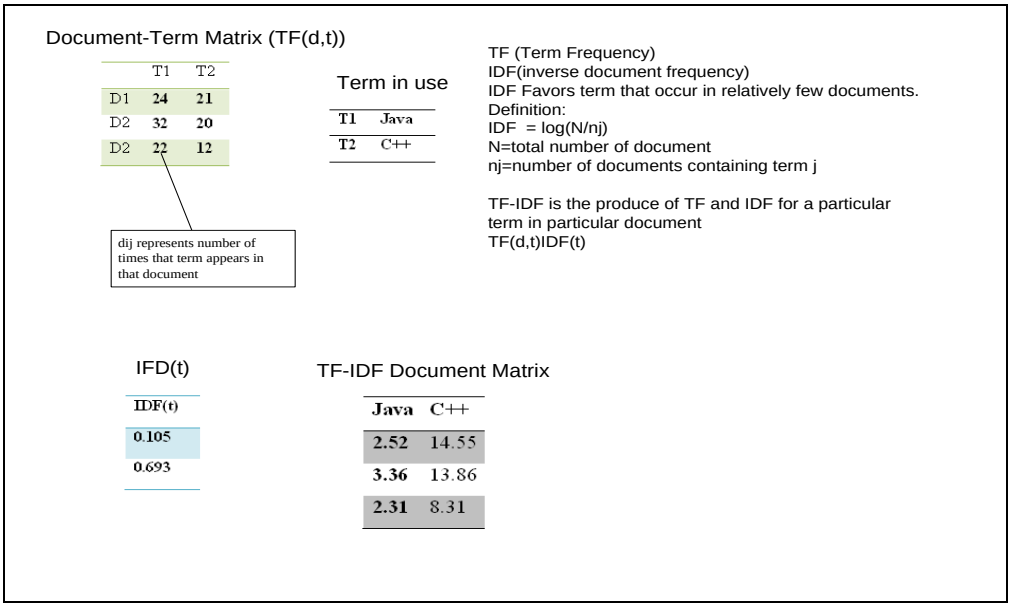


Figure 5.8: TF-IDF Example

Jaccard Similarity:

Matching will be greatly facilitated where the instances shared by two ontologies are identifiable. For example, if two classes share exactly the same set of instances, then these two classes represent a correct match. Moreover, when classes do not share the same set of instances, accurate matching based on the instances is achieved; for example, the names of authors of books do not have any reason to change. So if the names of authors of books are different, then these are most certainly not the same books.

The *similarity* of sets A and B is the ratio of the sizes of the intersection and union of A and B.

Input:

- two sets of objects: *A* and B

Similarity function:

- *Sim* (C_1, C2) = |S1 ∩S2|/|S1∪S2| = *Jaccard similarity*.
- threshold: t

Example: similarity of sets {10, 12, 31} and {10, 31, 44, 50} is 2/5.

Disjoint sets have a similarity of 0, and the similarity of a set with itself is 1.

Output:

- all pairs of objects a∈ S1, b ∈S2, such that
- sim(C_1,C_2) ≥ t

Jaccard similarity Matching (T_i)

{

find the nominee concept for T_i

For each nominee T_k discovered,

calculate the Jaccard $\frac{P(Ti \cap Tk)}{P(Ti \cup Tk)}$ For T_i and T_k

If the highest similarity degree > threshold,

Matching is discovered

Else

Matching is unsuccessful.

}

Figure 5.9: Jaccard Algorithm

Soundex:

Soundex is a phonetic algorithm that indexes English names by their sounds. In other words, it uses phonetics to compare words. The algorithm, which produces a string consisting of the first letter of the word followed by three digits, is used to identify strings written in a common description and their meaning is then determined. The algorithm encodes the letters as follows:

Remove all vowels;

- W, H, B, F, P, V encoded as 1;
- C, G, J, K, Q, S, X, Z encoded as 2;
- D, T encoded as 3;
- L encoded as 4;
- M, N encoded as 5;
- R encoded as 6.

Examples:

- Srivastava = S6i1a23a1a = S61231 = S612
- Shribasdaba = Sh6i1a23a1a = S61231 = S612.

Table 5.1 shows the result of applying string matching algorithms to a number of ontologies provided by OAEI (2005) and which are examined in detail in Chapter 6. The table shows two measurements (precision and recall) used to compute the

performance of algorithms, and the F-measure, which is displayed graphically in Figure 5.10.

$$Precision = \frac{Num\ of\ correct\ found\ match\ pairs}{Num\ of\ found\ matched\ pairs}$$

$$Recall = \frac{Num\ of\ correct\ found\ matched\ pairs}{Num\ of\ existing\ matched\ pairs}$$

$$F - Measure = \frac{2 * Precision * Rcall}{Precision + Rcall}$$

Table 5.1: String Similarity Evaluation

	Levenshtein		**Jaro**		**SoftTF/IDF**		**Jaccard**		**Soundex**	
Test	**Pre.**	**Rec.**	**Pre.**	**Rec.**	**Pre.**	**Rec.**	**Pre.**	**Rec.**	**Pre.**	**Rec.**
101	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
204	**0.96**	**0.96**	**0.82**	**0.80**	**0.97**	**0.97**	**0.96**	**0.92**	**0.85**	**0.84**
230	**0.80**	**0.78**	**0.87**	**0.78**	**0.90**	**0.78**	**0.81**	**0.55**	**0.87**	**0.78**
230	**0.60**	**0.64**	**0.36**	**0.66**	**0.72**	**0.64**	**0.36**	**0.66**	**0.66**	**0.66**
240	**0.76**	**0.81**	**0.62**	**0.83**	**0.75**	**0.83**	**0.75**	**0.83**	**0.67**	**0.83**
241	**0.97**	**0.94**	**0.80**	**0.81**	**0.97**	**0.94**	**0.92**	**0.94**	**0.97**	**0.94**
247	**0.79**	**0.90**	**0.86**	**0.78**	**0.91**	**0.96**	**0.86**	**0.78**	**0.93**	**0.96**
Overall	**0.84**	**0.86**	**0.76**	**0.81**	**0.89**	**0.87**	**0.81**	**0.81**	**0.85**	**0.86**
f- measure	**0.85**		**0.78**		**0.88**		**0.80**		**0.85**	

Table 5.1 shows comparison between different string match algorithms over several ontology tests. The results from the table show that Soft TF/IDF is much better than the others.

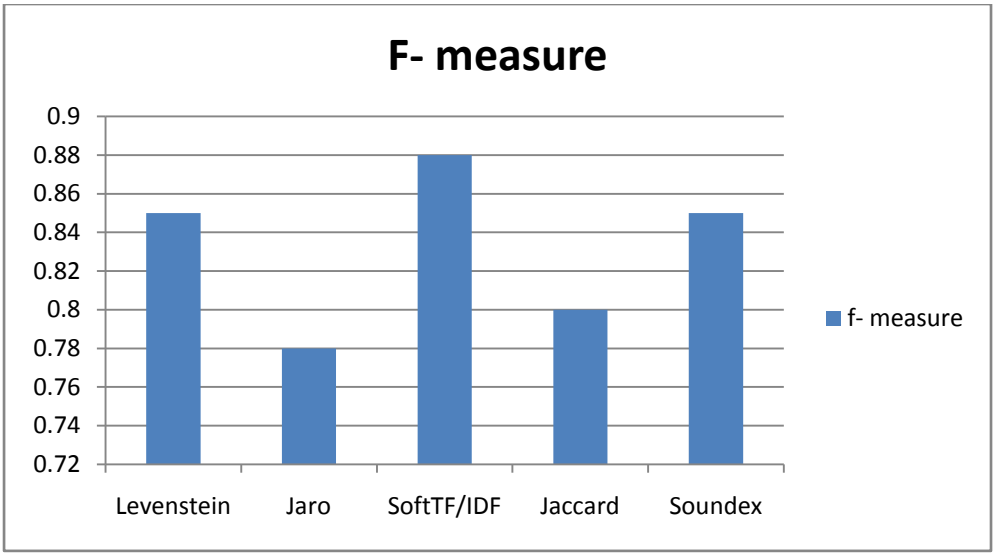


Figure 5.10: F-measure of the String Matchers

A weakness of string matching is that concepts such as Person and Human, which at first sight are equivalent, are not considered equivalent, as they are not superficially the same. A dictionary should be used in order to deal with this problem.

5.1.2.2 Linguistic Matching

The terminology used for naming and labelling concepts and properties is an important aspect of ontologies and provides information on the similarity between the ontology elements. However, linguistic features are also important for deriving an initial set of alignments to be refined by exploiting other kinds of matching. In fact, names of classes or properties are considered to provide one of the most important clues as to whether two terms are equal or not; therefore, this system tries to find relations between terms from different ontologies based on the details of their names. Such linguistic matching relies on algorithms and the use of external lexicon-based resources such as dictionaries, which are typically used to find close relationships such as synonymy between two terms and to compute the semantic distance between them in order to decide if a relationship holds.

This process is based on linguistic analysis. There are two general techniques for label matching, the first of which employs linguistic analysis steps, such as

abbreviations, avoiding recurrence and particle-ending. The other is matching the labels to determine the relationship between them.

In general, the linguistic similarity between terms is computed by considering labels and descriptions. Knowledge-based matchers take as input two concept (or synset) identifiers defined in WordNet and produce semantic relations by exploiting their structural properties. They are often based on either similarity or relatedness measures. If the value of the measure exceeds the given threshold, a certain semantic relation is produced. Otherwise, "Idk" (I don't know) is returned. This technique is implemented by using thesauri and WordNet, following an approach which is essentially the structural congruence between labels based on the hidden meanings of the words that they represent. WordNet, which takes two concept (synset) identifiers as input and returns the semantic relation holding between them, is considered not only to provide synonyms, hypernyms and hyponyms, but also to exploit additional structure to detect relationships between concepts (dinner→meal). For example, it considers synonyms as equivalent and hyponyms as subsumed, finding Match and Alignment to be similar classes (car→automobile).

Determine the similarity between concept C1 ∈ O1 and C2 ∈ O2 using the following steps:

Exact Match (Term (C1) = Term (C2))

Use WordNet to check if Synonyms (C1,C2) =1

Otherwise = 0

The results must be a value between 0 and 1 using a variant of the Jaro distance.

The similarity is computed by:

$$\frac{2 \times \sum_{i\ th\ token\ of\ C1, j\ th\ token\ of\ C2} \text{Jaro}(i,j)}{|C1| + |C2|}$$

where *C*1 is the first Class name, *C*2 is the second Class name, Jaro returns the string distance between the *i*th and *j*th token from the respective names, and |C1| and |C2| are the lengths of C1 and C2 respectively.

$$\sigma(s,t) = \begin{cases} 1 & \text{if } \sum(s) \cap \sum(t) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Figure 5.11: Using WordNet to Compute the Synonyms

Semantic relations can be defined in terms of senses as follows.

- *An equality relation* holds between two concepts when there is at least one sense of the first concept which is a synonym of the second.
- *The more general / specific relation* holds between two concepts when one is more general than the other.
- *A mismatch relation* holds between two concepts when they have no sense in common.

Words with possibly multiple meanings are connected together, out of which a single sense emerges, given in *word#pos#sense* form. For example, *car#n#3* refers to the third WordNet noun sense of *car*. Usually, synonymy exists between two words when they share a similar sense. This also implies that one word can be replaced by its synonym in a context without any loss of meaning. In practice, most words are not perfect replacements for their synonyms, i.e. they are near synonyms. While these relations cannot be considered incorrect, they are of little relevance for the domain ontology.

In our approach, use of WordNet synonyms is not enough. Therefore, this system proposes combining the use of semantic relations of WordNet with the structure of the WordNet hierarchy to find, for each element of taxonomy source, what elements of taxonomy target may be semantically close (those with shared generalisations in WordNet). This allows us to bring together, for example, "cantaloupe" and "watermelon", which are not synonymous but are two specialisations of the concept of melon.

WordNet::Similarity uses the WordNet database to calculate the semantic similarity between two words. It facilitates different measures and methods to calculate semantic similarity. For example, in the case of nouns, could be used relations of four types:

- hypernym: Y is a hypernym of X if every X is a (kind of) Y
- hyponym: Y is a hyponym of X if every Y is a (kind of) X

- holonym: Y is a holonym of X if X is a part of Y
- meronym: Y is a meronym of X if Y is a part of X.

In using a WordNet-based matcher the (lexical) relations which are provided by WordNet should be translated to logical relations [26, 94], based on the following rules:

- $A \subseteq B$, if *A* is a hyponym or meronym of *B*. For example, *author* is a hyponym of *creator*, therefore deducing that author $\subseteq$ creator.
- $A \supseteq B$, if *A* is a hypernym or holonym of *B*. For example, *Asia* is a holonym of *Jordan*, therefore deducing that Asia $\supseteq$ Jordan.
- $A = B$, if A and B are connected by a synonymous relation or they belong to one synset. For example, *quantity* and *amount* are synonyms, therefore deducing that quantity = amount.
- $A \perp B$, if A and B are connected by antonymy relations or are siblings in a *part of* hierarchy. For example, *Jordan* and *Syria* are siblings in the WordNet *part of* hierarchy, therefore deducing that Jordan $\perp$ Syria.

Table 5.2: Semantic Relation Produced by the WordNet

Concept 1	Concept 2	Semantic Relation
car	automobile	$\equiv$
blue	yellow	Idk
tail	cat	$\subseteq$
car	minivan	$\supseteq$

Table 5.2 shows the semantic relations that could be held between concepts when using WordNet as external information. Therefore these relations could be equality relation, more general / specific relation or mismatch relation.

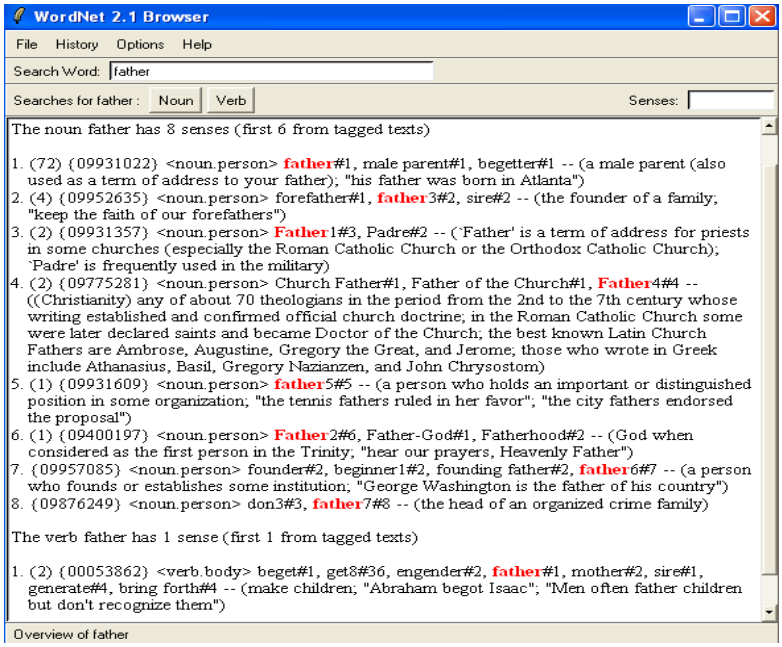


Figure 5.12: Snapshot of WordNet Result

The similarity measure of Leacock and Chodorow is based on the shortest path length between two concepts in an "is-a" hierarchy and *scale* the path length by the overall depth *D* of the taxonomy.

Example:

Input: two concepts (Wolf, Dog)

Function: $Sim_{lch}(c_1, c_2) = -\log \frac{ShortestLen(c_1,c_2)}{2*TaxonomyDept\ h}$

Output: Similarity(wolf, dog) = 0.60

Figure 5.13 illustrates the main classes of linguistic matcher.

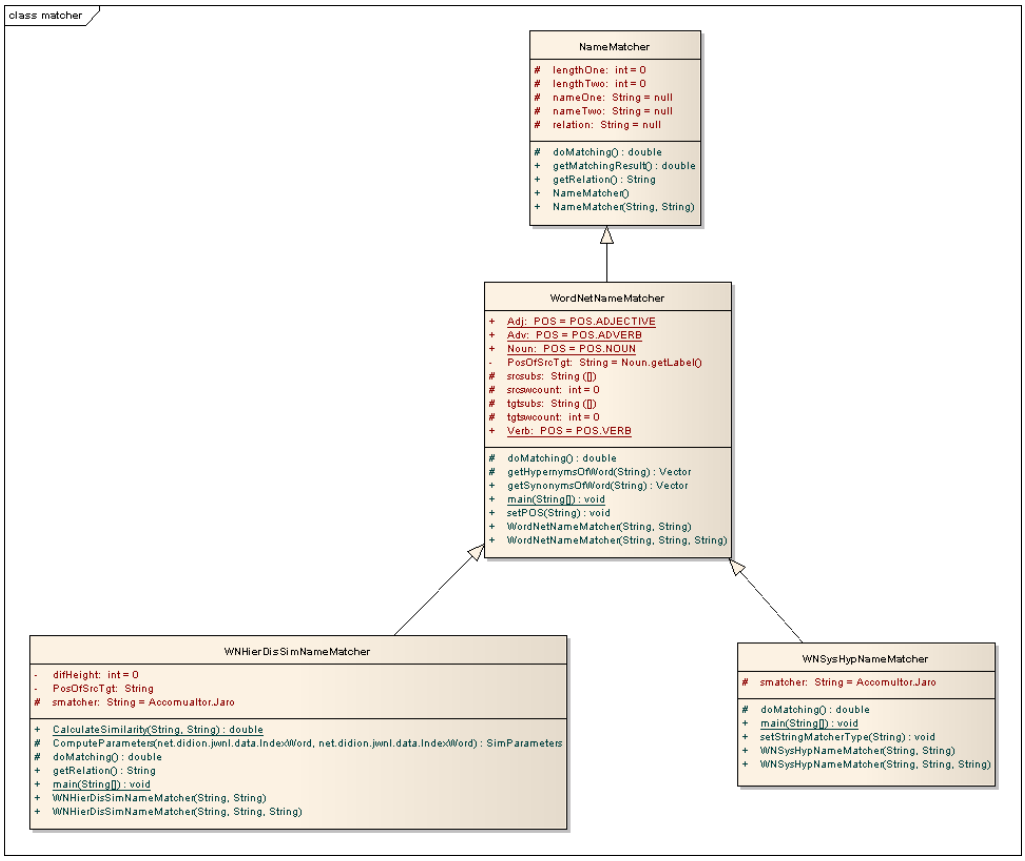


Figure 5.13: Linguistic Matcher Classes

5.1.2.3 Structure Matching

The aim of structural matching is to link an element of source taxonomy with an element of target taxonomy. The mappings generated are mainly specialisation matches, based on calculations of the similarity of the source element with all those under the target taxonomy. Indeed, this kind of matching depends on what are considered the most important features of ontology nodes (e.g. class: super-classes and Sub-class, property: super properties and sub properties). Therefore, similarity is based on the nodes of graphs.

The similarities between two concepts can be obtained from the language and from real attributes; and not only the similarities between the descriptions of their components, but also from similarities between the structures of the graphs representing them. The internal structure of similarities can be obtained by calculating the number of similar attributes divided by the attributes of a class.

Taxonomy is generally represented by an acyclic graph whose nodes are concepts and arcs corresponding to linked subclasses. Class inheritance analysis (is-a) considers the hierarchical connection between classes in order to identify "is-a" relationships.

Taxonomy (C, HC) includes a set of concepts C and a hierarchy subsumption between concepts HC. A concept is defined by its label and subclass relationships, which connect to other concepts. The label is a name (string) which describes entities in natural language and can be an expression composed of several words. Subclass relations establish links between concepts.

The intuition behind the algorithm is that if two concepts lie in similar positions with respect to *is-a* hierarchy relative to concepts already aligned in the two ontologies, then they are likely to be similar as well. It is important to mention here two important rules that help to ensure correct matching. First, if the super-concepts of two classes are the same, then these two concepts may be similar to each other. The second rule is that if the sub-concepts of two classes are the same, it is likely to say that the concepts are also similar.

In terms of the semantic matcher, this system combined several rules and techniques, as follows.

Structural analysis identifies identical classes by looking at their attributes and related (linked) classes. The main idea is that two classes of two ontologies are similar or identical if they have the same attributes and the same neighbour classes. Hence, matching concepts are based on structural similarity with regard to class hierarchy.

Our assumption is that a combination of features and similarity measures leads to better alignment results, compared with using only one at a time. This approach is used to find semantically similar ontologies based on their structural information.

In terms of internal structure, it can be recognised that it is based on the internal structure of entities, which can be used to calculate the similarity between them. Therefore, many criteria are used such as attributes and relations, their cardinality and the transitivity and/or symmetry of their properties. This method is thus commonly

combined with others such as terminological, structural or extensional ones, in order to reduce the number of alignment candidates. Moreover, some general ideas have been used based on the position of entities within their hierarchies. Hence, if two entities from two ontologies are similar, this suggests that their neighbours might also be somehow similar. Therefore, in general, in deciding whether two entities are similar, certain criteria should hold:

- If their direct super-entities are similar, then they are similar;
- If their sibling-entities are similar, then they are similar;
- If their direct sub-entities are similar, then they are similar;
- If most of their descendant-entities are similar, then they are similar;
- If most of their individuals are similar, then they are similar;
- If the domain and range of two properties are equal, then the properties are also similar.

In order to deal with these rules, an arbitrary numeric value to each of these results is assigned; thus the same property name is given the same value, domain and same range, and different values to different property names, domains and ranges.

In order to match concepts between two taxonomies, a well-defined measure to facilitate the evaluation of our system is applied. In addition, it allows us to leverage special-purpose techniques for the matching process. In fact, our work in structure matching based on [82] with changing the lexical matching in order to improve the result.

A general rule is maintained that helps us in this kind of matching, whereby two concepts are similar only if:

- Their sub-concepts are the same. Or
- Their super-concepts are the same.

The similarity between the sub-concepts or super-concepts of two concepts *e*1 and *e*2 is defined as follows:

$$Sim_{Supc\ /Subc}\ (e1, e2) = \frac{\sum_{i=1}^{n}(\sum_{j=1}^{m} sim\ (e1_{hi}\ ,\ e2_{hj}))}{(n+m)/2}$$

where

hi is the sub-concepts or super-concept I of eI

hj is the sub-concepts or super concept j of e2

n is the number of super-concepts or sub-concepts of eI

m is the number of super-concepts or sub-concepts of e2.

Because it is a matrix, the following formula should be used in order to normalise the result:

$$Sim_{Taxo}\ (e1, e2) = W_{subc}\ \times Sim_{subc}\ (e1, e2) + W_{supc}\ \times Sim_{supc}\ (e1, e2)$$

where

W_{subc} and W_{supc} are the weights which indicate respectively the importance of the similarity methods.

Sim_{subc} and Sim_{supc} are that such $W_{subc} + W_{supc} = 1.00$.

Figure 5.14: Taxonomy Matcher

In this kind of matching, a general rule is maintained, which states that any two concepts are equal if their properties are equal. In order to compute the similarity between two concepts *e*1 and *e*2 based on their structures, the following formula is used (figure 5.15):

$$Sim_{stru}(e,f) = \frac{\sum_{i=1}^{n}\left(\sum_{j=1}^{m} sim_{prop}(e_{pi}, f_{pj})\right)}{(n+m)/2}$$

where

pi is property i of e1.

pj is property j of e2.

n is the number of properties of e1.

m is the number of properties of e2.

In order to compute the similarity of two concepts based on the structure of their properties, the following two rules should be followed. First, it is possible to say that two properties are equal only if:

They have the same name, and /or

The domains and the ranges of the two properties are equal.

Secondly, the similarity between two concepts e_{pi} and f_{pj} is given by:

$$Sim_{prop}(e_{pi}, f_{pj}) = \max\left(Sim_{lexi}(e_{pi}, f_{pj}), \frac{Sim(e_{pi\,dom}, f_{pj\,dom}) + Sim(e_{pi\,ran}, f_{pj\,ran})}{2}\right)$$

where

pi^{dom}, pi^{ran} are the domain and the range of the properties pi and

pj^{dom}, pj^{ran} are the domain and the range of the properties pj.

Moreover, in some description languages such as OWL, some properties can be used such as "sameIndividualAs" or "same-ClassAs" to explicate that two entities are the same.

Jaro – Winkler is used as lexical matcher and it formula as follow:

$$Jaro - Winkler(s,t) = Jaro(s,t) + \frac{P'}{10} * (1 - Jaro(s,t))$$

Where P is the longest common prefix of s and t, and let P'= max (P, n) where n is integer number.

Example:

Jaro (JUNTHA, JUNHTA) = 1/3 * (5/5 + 5/5 + (5 - 1)/5) = 0.944

Three initial characters match, JUN, for a Jaro-Winkler distance

Jaro-Winkler (JUNTHA, JUNHTA) = 1 + 0.1 * 3 * (1.0 - 0.944) = 0.961

Figure 5.15: Semantic Matcher

Figure 5.16 illustrate the main classes of structure matcher.

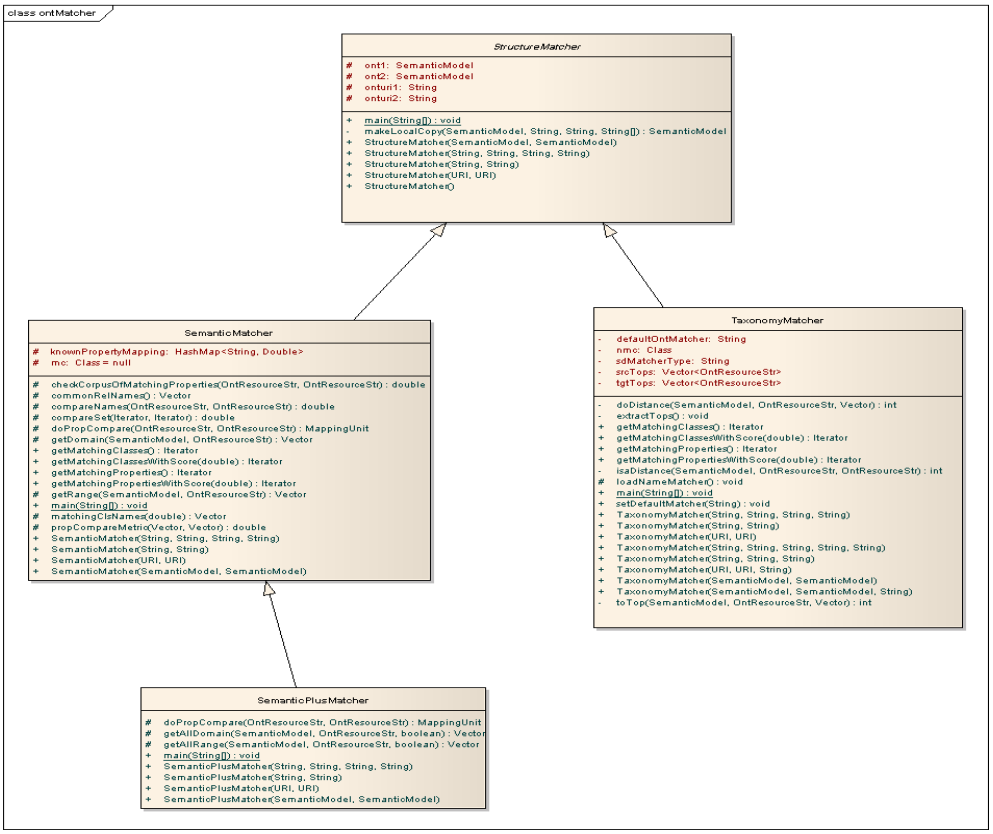


Figure 5.16: Structure Matching Classes

5.1.2.4 Heuristic-based Strategies

This phase of our system uses several features of ontologies (i.e. their structure, definitions of concepts and instances of classes) in order to find matches. Based on the idea that labelling is important and helps to align most of the entities, the structure also provides valuable help in identifying alignments. This step was developed this based on these two elements.

It considers the entities and structure of an ontology, i.e. class (C), property (P), relation (R), instance (I) and super-class (S). The distances between the input structures are then expressed in a set of equations. As described on figure 5.17 and 5.18.

$$Tot_{Sim(c,c')} = w * Sim_C(c,c') + w * Sim_P(c,c') + w * Sim_R(c,c') + w * Sim_I(c,c') + w * Sim_S(c,c')$$

where

$Sim_C(c, c')$ is the similarity between labels of classes,

$Sim_P(c, c')$ is the similarity between properties of classes,

$Sim_R(c, c')$ is the similarity between relations of classes,

$Sim_I(c, c')$ is the similarity between instances of classes,

$Sim_S(c, c')$ is the similarity between super-classes of classes,

w is the weight, which is set at $1/5$ in order to obtain results in the range $[0,1]$,

$Tot_{Sim(c,c')}$ is the average of all of these similarities, in the range $[0,1]$.

Figure 5.17: Heuristic Matcher Equation

The following is the function of heuristic match:

```
Function heuristicMatch (Ontology o1, Ontology o2) {
  for (All concept pairs (c, c') where c ϵ o1 and c' ϵ o2) {
Sim_C = ComputeNameSimilarity (c, c')
Sim_P = (W* findCommonAttributes (c, c')) + (W* matchDataTypes (c, c')) + (W * matchDataInstance (c, c'))
Sim_R = (W * findRelationship (c, c')) + (W* matchNameRelationship(c, c'))
Sim_I = (W* findCommonInstance (c, c') + (W* matchInctance (c, c'))
Sim_S = W* ComputeNameSuperClass (c, c')
    //compute overall similarity
      Tot_Sim(c,c') = w * Sim_C(c,c') + w * Sim_P(c,c') + w * Sim_R(c,c') + w
                      * Sim_I(c,c') + w * Sim_S(c,c')
    }//end for
}//function heuristicMatch
```

Figure 5.18: Heuristic Matcher Function

The output is one-to-one or one-to-many correspondences. This strategy is based on string similarity (i.e., Monge-Elkan metric [15]) structure and instances.

Monge-Elkan distance is recursive matching scheme for comparing two long strings s and t. By assuming that the strings s and t are broken into substrings (tokens), i.e., s = $s_1 \ldots s_K$ and t = $t_1 \ldots t_L$. The intuition behind this measure is the assumption that s_i in

s corresponds to a t_j with which it has highest similarity. The similarity between s and t equals the mean of these maximum scores.

$$Monge - Elkan(A,B) = \frac{1}{|A|} * \sum_{i=1}^{|A|} \max_{J=1}^{|B|} match\ (A_i, B_j)$$

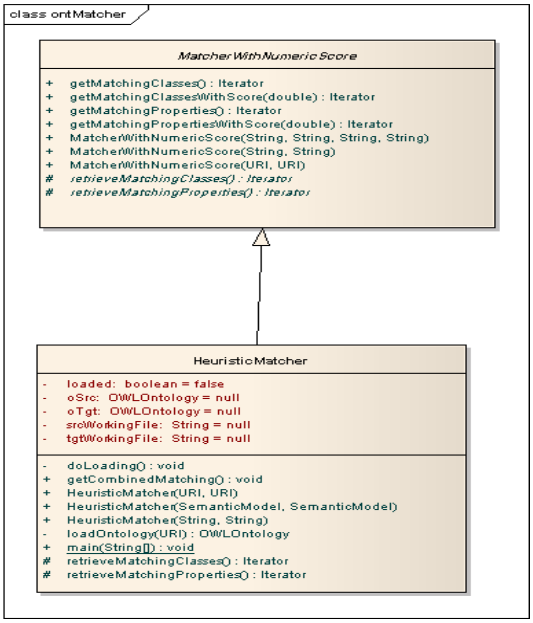


Figure 5.19: Heuristic Matcher Class

Indeed, the results from any matcher will be a pair of matching ontological entities with either numeric values representing similarity or symbolic constructors specifying the relationships, e.g. subsumed, subsuming, disjoint with. So *MatcherWithNumericScore* returns numeric similarities.

Figure 5.20 shows a simplified representation of two ontologies of cars, A and B, while Figure 5.21 depicts an assessment of the similarity between Ontology A and Ontology B.

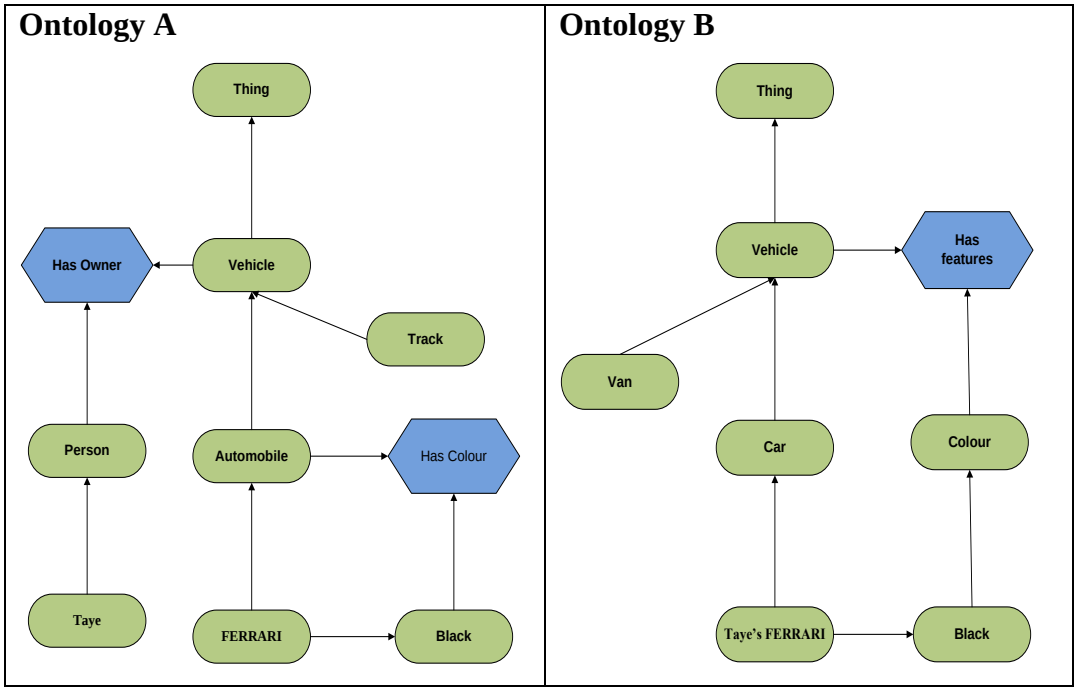


Figure 5.20: Two Car Ontologies

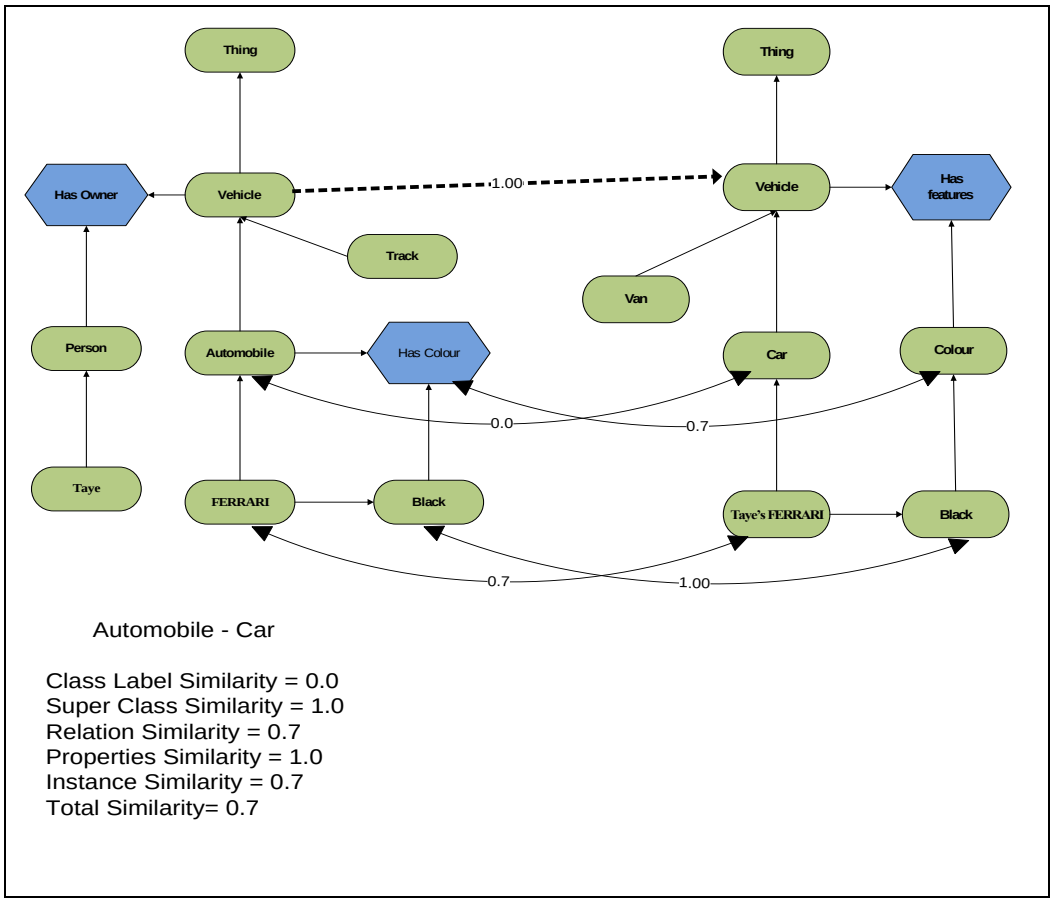


Figure 5.21: Assessing the Similarity between Ontology A & Ontology B

In heuristic matching, iteration is one of the most important steps in ontology alignment, which takes into account the structure of the input ontologies. It enables the process to be repeated several times, by generating and updating the assessed similarities.

The sigmoid strategy combines multiple results using a sigmoid function, which is a smoothed threshold function, showing the importance of retaining high individual prediction values and removing low ones.

This technique starts after the application of the normalisation process on the input elements, then compares class and property names in terms of editing distance and substring distance between entity names. The algorithms next create a distance matrix in order to determine the alignment group from the distance.

This algorithm is used in order to cover most possible features of ontologies (i.e. terminological, structural and extensional); on the other hand, this explicates recursive relationships and tries to find the best matches through iteration. In general, this method faces problems when the viewpoints of two ontologies are highly different; thus, in order to achieve a high quality result, several of the above criteria should be combined, so that the rules which can be applied here are:

- Any two concepts are probably the same if their **labels** are the same.
- Any two concepts are equal if their **properties** are equal, even if their labels are different.
- Two concepts that have the same **instances** are the same.

5.1.3 Aggregation

The results discussed here have been calculated using string, linguistic, structure and heuristic matchers. Indeed, with several matching strategies/ algorithms, there are several similarity values for a candidate matching (e1; e2). Therefore, combining them is an effective way to achieve high accuracy for a larger variety of ontologies,

so this step extracts the combined matching result from the individual matcher results stored in the similarity matrix. For each combination of ontology entities, the strategy-specific similarity values are aggregated into a combined similarity value, for example, by taking the maximum value.

Using threshold is the easiest way to select the matching results. Such threshold-based filtering allows us to retain only the most similar entity pairs. For the combination of the match results, the average value has been computed and a selection has been made using a threshold, for example.

$$Semantic\ Distance(C, C^{'}) \leq Threshold$$

5.1.3.1 Alignment Aggregator

The alignment aggregator initialises an aggregator containing matching results from different matchers. It defines a list of methods to be implemented by any actual aggregators, among which it returns the results of the aggregation and initialising matrix (String id, int m, int n) to create an $m * n$ matrix with name *id*. It then computes the weighted average of results returned by different matchers with regard to each pair of concepts/ properties from the source and target ontologies.

Alignment Matrix defines the data structure to hold matching results from different matchers. An instance of Alignment Matrix is illustrated in Figure 5.22, with each row corresponding to a pair of entities from the source and target ontologies, and each column corresponding to a particular matcher. Each row of the resultant matrix is assigned a symbolic name for easy access. This method returns the names of rows (matching entity pairs) as an array of strings.

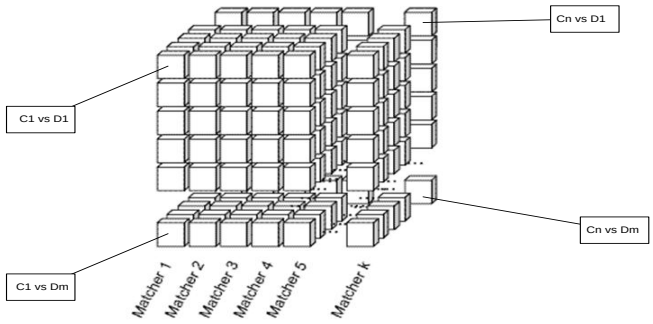


Figure 5.22: Alignment Matrix

Relations among ontological entities might not always be equivalent. This class takes into account different types of relations, and computes the weighted average only for results with the same type of relation. It distinguishes three types of relations, namely *more general than* (>), *more specific than* (<) and equivalent (=).

Generally, similarity aggregation can be expressed through:

$$Sim_{agg}(e,f) = agg(sim_1(e, f),... sim_k(e, f))$$

where (e, f) is a candidate alignment and *agg* a function over the individual similarity measures sim_1 to sim_k. This function often leads to a simpler equation:

$$Sim_{aggs}(e,f)_= \frac{\sum_{k=1..n} W_k \times ad_{jk}(sim_k(e,f))}{\sum_{k=1..n} W_k}$$

where W_k is the weight for each individual similarity measure and adj_k is an adjustment function to transform the original similarity value ($adj : [0,1] \rightarrow [0,1]$). An explanation of the use of these variables will follow.

For this aggregation, the weights W_k have to be determined. They are assigned manually or by learning, e.g. by using machine learning on a training set. All the individual weights are set to 1, the adjustment function adj_k is set to be the identity function *id* and the value is not changed. The result is a simple average over all individual similarities:

$$W_k = 1, adj_k(x) = id(x)$$

In this approach, this system is looking for similarity values only, supporting the claim that two entities are equal. A missing similarity is not necessarily treated as negative evidence. Linear summation is most widely used in related approaches, in most of which the user himself assigns the weights.

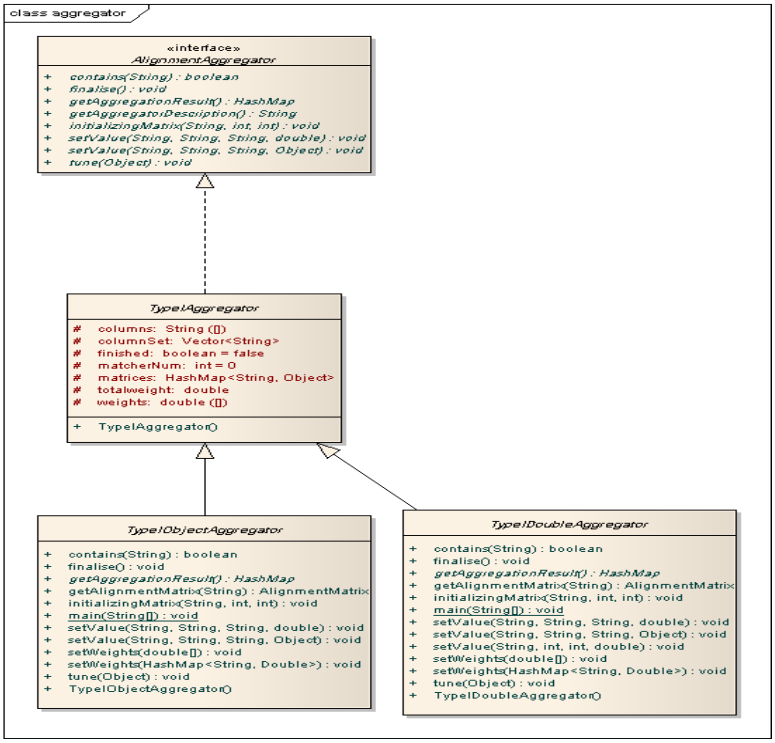


Figure 5.23: Alignment aggregation classes

AlignmentAggregator interface defines a list of methods to be implemented by any concrete aggregators, among which *getAggregationResult()* returns the results of aggregation and *initialisingMatrix* (String id, int m, int n) constructs an $m \times n$ matrix with name *id*.

TypeIDoubleAggregator and *TypeIObjectAggregator* initialise an aggregator containing double-typed and *MatchingUnit* similarities from different matchers.

5.1.3.2 Setting Weights

The description assigns to each matcher a corresponding weight specified in the array as input weights. For the running example, simple linear aggregation is used.

Weighted: This strategy determines a weighted sum of similarity values of the individual matchers and needs relative weights, which should correspond to the expected importance of the matchers. It overcomes the drawbacks of the *Average* strategy by assigning relative weights to individual matchers.

Average: This strategy represents a special case of the weighted strategy and returns the average similarity over all individual matchers, so it considers the individual similarities to be equally important and cannot distinguish between them.

Ten individual similarities are assumed when comparing two concepts, of which only the first are shown in the following formula:

Weighted average [28] Fuzzy aggregation operators are used for assimilating homogeneous quantities in a way that preserves the structure of the aggregated domains.

Let O be a set of objects which can be analysed in n dimensions. The weighted average between two such objects is as follows:

$$\forall s, s' \in O, \delta(s, s') = \frac{\sum_{i=1}^{k} wi \times \delta(si, si')}{\sum_{i=1}^{k} wi}$$

such that δ(si, si') is the dissimilarity of the pair of objects along the ith dimension and wi is the weight of dimension i.

A simple average function is a function such that all weights are equal. If the values are normalised, the weighted average is normalised. In fact, the normalised weighted sum is also a weighted average.

Sim_{agg}(ol:human,o2:person) = (1.0 * sim_{name}(ol: human,o2: person) + 1.0 *$sim_{SuperConcept}$(ol: human,o2: person) + 1.0 * $sim_{Relation}$(ol: human,o2: person) + 1.0 * $sim_{Teaxonmy}$(ol: human,o2: person) + .. •) /10 = 0.5.

In the process, the aggregation of many methods is performed once per candidate alignment and is therefore critical for the overall efficiency. Therefore, the following function is performed with a manually assigned weight in this step; this case will set the weight automatically to 1.00.

The description computes the weighted average of results returned by different matchers with regard to each pair of concepts/ properties from the source and target ontologies.

In general, assigning weight to each matching algorithm is one of the hardest and most important topics in ontology alignment; indeed, the good question is how to assign a correctness weight to each specific matcher. Actually, one answer to this question could be by giving the best match a weight of 1, and the second best matcher 0.80. For example, the similarity in name class is more significant than similarity in comments; therefore, it could be assigned high weight value to name matcher then the weight value to comment matcher. The another answer requires an expert who could suggest manually setting the weights value by using experience numbers which are based on experience. On the other hand, in order to solve the problem of the manual parameter setting, it is possible to use machine learning based parameter optimisation. Our system used the average function which assigned equal weights for all matchers.

$$Sim_{overall\ (C1,D1)} = \frac{\sum_{i=0}^{n} w_{matc\ her\,(i)} * sim_{matc\ her\,(i)}(C1,D1)}{\sum_{i=0}^{n} w_{matc\ her\,(i)}}$$

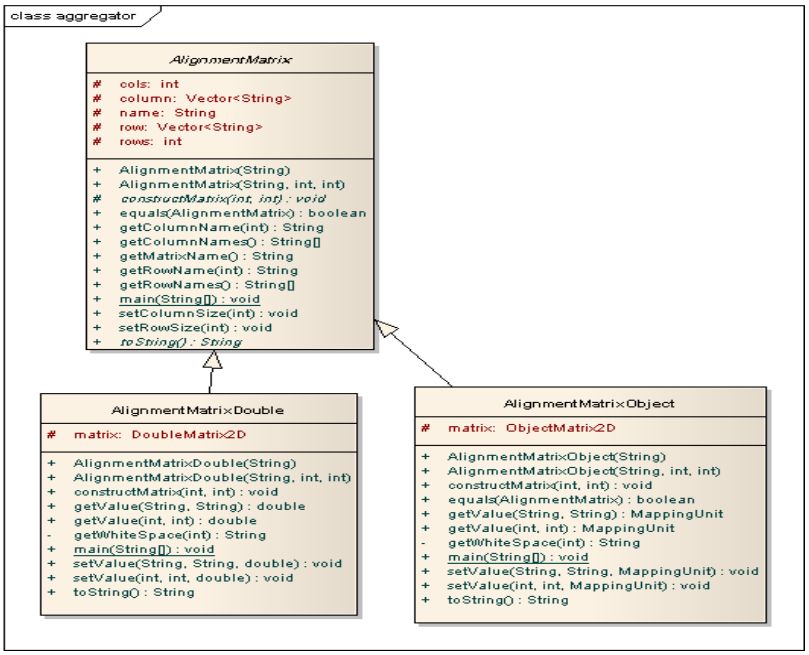


Figure 5.24: Alignment Matrix Classes

AlignmentMatrix defines the data structure to hold matching results from different matchers. An instance of *AlignmentMatrix* is each row corresponds to a pair of entities from the source and target ontologies and column corresponds to a particular matcher.

AlignmentMatrixDouble and AlignmentMatrixObject define a double-typed matrix and MacthingUnit respectively.

5.1.4 Output

The output of the alignment algorithm is a group of alignment relationships holding between terms from the source ontologies. The similarity values range between [0..1] that indicate how well the corresponding entities in O1 match their counterparts in O2.

- If $Sim(e1, e2) = 1$ then R is an equivalence (=) relation.
- If $Sim(e1, e2) = 0$ then R is a disjointedness ($\perp$) relation.
- If $Sim(e1, e2) > t$ (threshold) then R is a subsumption ($\supseteq$ *or* $\subseteq$) relation.

5.1.4.1 Output Format

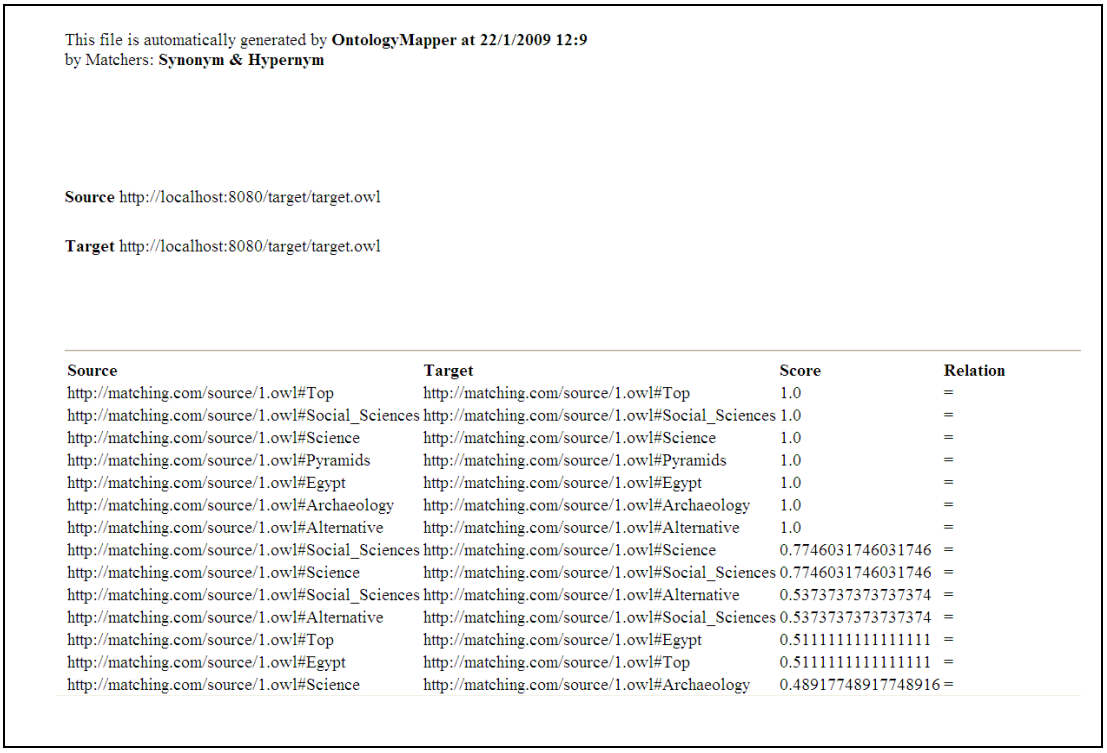
In fact, our current system finds one-to-one or one-to-many mapping with equivalence relations between classes and properties in ontologies, as well as more complex relations, such as broader and narrower. This is achieved by converting equivalence relations to broader and narrower relations by identifying the hierarchical relationships between classes in two ontologies.

The output format specifies methods for retrieving all matching classes or properties and those matching classes or properties with similarity values greater than a predefined threshold. The output from ontology matching algorithms is a pair of matching ontological entities with numeric values representing similarity and with symbolic constructors specifying the abstract relationships, e.g. *equal* and *subsuming*.

This output is provided in three types of format: HTML, XML and OWL. In fact, regarding output format, the system used format such as the format used in API and CROSI Mapping System.

HTML Format

```
<TD> URI of the first entity </TD>
<TD> URI of the second entity <TD>
<TD> rel </TD>
<TD> n </TD>
```



This file is automatically generated by **OntologyMapper at 22/1/2009 12:9**
by Matchers: **Synonym & Hypernym**

Source http://localhost:8080/target/target.owl

Target http://localhost:8080/target/target.owl

Source	Target	Score	Relation
http://matching.com/source/1.owl#Top	http://matching.com/source/1.owl#Top	1.0	=
http://matching.com/source/1.owl#Social_Sciences	http://matching.com/source/1.owl#Social_Sciences	1.0	=
http://matching.com/source/1.owl#Science	http://matching.com/source/1.owl#Science	1.0	=
http://matching.com/source/1.owl#Pyramids	http://matching.com/source/1.owl#Pyramids	1.0	=
http://matching.com/source/1.owl#Egypt	http://matching.com/source/1.owl#Egypt	1.0	=
http://matching.com/source/1.owl#Archaeology	http://matching.com/source/1.owl#Archaeology	1.0	=
http://matching.com/source/1.owl#Alternative	http://matching.com/source/1.owl#Alternative	1.0	=
http://matching.com/source/1.owl#Social_Sciences	http://matching.com/source/1.owl#Science	0.7746031746031746	=
http://matching.com/source/1.owl#Science	http://matching.com/source/1.owl#Social_Sciences	0.7746031746031746	=
http://matching.com/source/1.owl#Social_Sciences	http://matching.com/source/1.owl#Alternative	0.5373737373737374	=
http://matching.com/source/1.owl#Alternative	http://matching.com/source/1.owl#Social_Sciences	0.5373737373737374	=
http://matching.com/source/1.owl#Top	http://matching.com/source/1.owl#Egypt	0.5111111111111111	=
http://matching.com/source/1.owl#Egypt	http://matching.com/source/1.owl#Top	0.5111111111111111	=
http://matching.com/source/1.owl#Science	http://matching.com/source/1.owl#Archaeology	0.48917748917748916	=

Figure 5.25: HTML format result

XML Format

```
< Cell>
 < entity1 rdf:resource="(URI of the first entity)"  >
 < entity2 rdf:resource="(URI of the second entity)">
<measure rdf:datatype="xsd:float"> n </measure>
 <relation> rel </relation>
</Cell>
```

Figure 5.26: XML format result

OWL Format

Matching results are saved in OWL format using ⟨*owl* : *sameAs*⟩. See Appendix A for more result.

```
<owl:Class rdf:about="(URI of the first entity)">
<owl:sameAs>
<owl:Class rdf:about="URI of the second entity"/>
</owl:sameAs>
</owl:Class>
```

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  </owl:Class>
  <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasFather">
    <owl:sameAs>
      <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasFather"/>
    </owl:sameAs>
  </owl:Class>
  <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#shoesize">
    <owl:sameAs>
      <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#shoesize"/>
    </owl:sameAs>
 </owl:Class>
  <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Woman">
    <owl:sameAs>
      <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Woman"/>
    </owl:sameAs>
  </owl:Class>
  <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Person">
    <owl:sameAs>
      <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Person"/>
    </owl:sameAs>
  </owl:Class>
  <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Man">
    <owl:sameAs>
      <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Man"/>
    </owl:sameAs>
  </owl:Class>
  <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasSpouse">
    <owl:sameAs>
      <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasSpouse"/>
    </owl:sameAs>
```

Figure 5.27: OWL format result

5.2 Summary

The major steps in the approach include: pre-matching, matching process and output. A new ontology alignment approach has been proposed based on four kinds of strategy:

String-based strategies: a well-known algorithm is used to compute an optimal one-to-one mapping. Our system considers the labels in the ontologies to be very important features, as is any additional information like comments or language elements, since comments in ontologies can work very well when the ontologies are created in a well controlled environment with strong academic background such as a university or research institution.

The WordNet dictionary: (a linguistic-based matcher) was used in order to calculate similarities between words for the comparison of node names in ontologies.

Structure-based matching: The results from this matcher were calculated from a detailed analysis of graphs, using many features of ontology such as taxonomy (super-class and subclass relations). Many rules were also applied to the graphs.

Heuristic-based matching: This algorithm seeks matches by analysing the structural similarity between the ontologies and combines this with label similarity measures to produce matching correspondences. The most important feature of ontology is labels, which could alone return very satisfying results.

In general, in order to discover matching, multiple matching algorithms based on several similarity measures should be executed (such as names, structure or external information). The main task of these algorithms is to determine similarity values between candidate mappings (e1, e2) based on their definitions in source ontology and target ontology (O1, O2) respectively. Each matcher determines an intermediate matching or alignment result for each possible candidate matching with similarity value between [0...1]. The result of the matching execution phase with k matching algorithms, m entities in O1 and n entities in O2 is a $k \times m \times n$ matrix of similarity values, which is stored in the repository for later combination and selection steps. The value of each matrix entry specifies the similarity of the specific pair to which the entry corresponds. The match result from the previous step can be aggregated into a single similarity value for two ontologies, called combined similarity. This depends on the chosen matchers and their combination strategy.

Finally, the output can be in three formats: OWL, XML or HTML.

Chapter 6 Evaluation

It can be argued that the most significant and crucial issue when suggesting a new approach is its evaluation. Therefore, this chapter presents many test cases which are used to evaluate the performance of our system in different scenarios, followed by the experimental methodology, test data sets and results. Our system also is compared with top-ranked systems on the Benchmark Test in the OAEI Campaign 2007. Finally, a case study is reported.

6.1 Ontology Alignment Evaluation Initiative

The OAEI is a coordinated international initiative to establish agreement for evaluating and improving the available ontology alignment techniques. The OAEI ontology matching campaign is a contest organised annually since 2004, comprising several kinds of tests, processes and measures for assessing the results.

The goals of the Ontology Alignment Evaluation Initiative are:

- assessing strengths and weaknesses of alignment systems;
- comparing the performance of techniques;
- increasing communication among algorithm developers;
- improving evaluation techniques;
- helping to improve the standard of ontology alignment/matching.

The OAEI addresses its goals by organising an annual evaluation event and by publishing the tests and results of the event for further analysis.

6.2 Types of Evaluation

Currently, there are many ontology matching systems that have been developed based on different strategies for various purposes. In order to evaluate their performance and their qualities, we focused on OAEI evaluation which employs a systematic approach to evaluate ontology matching algorithms and identify their strengths and weaknesses.

6.3 Evaluation Measures

One of the difficult tasks is selecting measures to evaluate a new approach. Therefore, standard measures which have been used in the related field of information retrieval [23, 97] are applied. In order to evaluate ontology alignment from the perspective of different requirements, different kinds of measures are needed [23, 97], which are confronted with the ontologies to be matched and their output compared to that of a reference alignment system. This reference is, in fact, not always available, useful or even subject to agreement, but is adopted only for the purpose of benchmarking.

6.3.1 Compliance Measures

In order to assess the different approaches or evaluate the degree of compliance of the results of matching algorithms, standard information retrieval metrics are used, presenting four values which are widely used to estimate the quality of the alignment process and its results: *precision, recall, overall* and *F-measure*.

There are many ways to evaluate qualitatively the results of an alignment process. One common possibility consists of offering a reference alignment *R* to which the result of the candidate matching algorithm *A* is compared. This process, using well-known criteria like precision and recall, adapted from information retrieval [23, 30, 97] for use in ontology matching [17], effectively compares which correspondences are discovered and which are not. These criteria are well understood and widely accepted.

6.3.1.1 Precision

Precision measures the number of correct mappings found against the total number of the set of alignments obtained by a certain method.

Given a reference alignment *R*, the precision of some alignment *A* is a function

$P : \Lambda \times \Lambda \rightarrow [0, 1]$ such that:

$$Precision = \frac{number\ of\ correct\ alignments\ found}{number\ of\ alignments\ retrieved\ by\ a\ certain\ method} \times 100\%$$

Alternatively, it can be defined as follows:

$$Precision = \frac{|A \cap R|}{|A|}$$

where *R* is the set of reference alignments and *A* is the set of alignments obtained by a certain method.

A precision of value 1 indicates that all alignments found are correct, but it does not mean that all alignments have been found. Therefore, precision must be balanced against the recall measure.

6.3.1.2 Recall

Recall measures the number of correct mappings found against the total number of existing mappings.

Given a reference alignment *R*, the recall of some alignment *A* is a function

$R: \Lambda \times \Lambda \rightarrow [0, 1]$ such that:

$$Recall = \frac{number\ of\ correct\ alignments\ found}{number\ of\ existing\ alignments} \times 100\%$$

Alternatively, it can be defined as follows:

$$Recall = \frac{|R \cap A|}{|R|}$$

A high recall value indicates that many of the alignments have actually been found, but it gives no information about the number of false alignments.

6.3.1.3 F-measure

The F-measure combines the measures of precision and recall as single efficiency measure.

$$F - measure = \frac{2 \times precision \times recall}{precision + recall} \times 100\%$$

6.3.1.4 Overall

The overall measure is the ratio of the number of errors to the size of the expected alignment. This measure is considered to represent the edit distance between an alignment and a reference alignment in which the only operation is 'error correction'. For that reason, it is considered a measure of the effort required to fix the alignment.

Given a reference alignment *R*, the overall measure of some alignment *A* is a function $O : \mathcal{A} \times \mathcal{A} \rightarrow [-1, 1]$ such that:

$$O(A, R) = R(A, R) \times (2 - \frac{1}{P(A, R)})$$

Alternatively, it can be defined as follows:

$$O(A, R) = 1 - \frac{|(A \cup R) - (A \cap R)|}{|R|} = 1 - \frac{|R - A| + |A - R|}{|R|}$$

6.4 Results and Discussion

In order to evaluate this system we check the performance by using all individual similarities, i.e. string similarity, linguistic similarity, heuristic similarity and structural similarity, in order to compute the correspondence between two elements from a different perspective. Also, comments about each matcher are mentioned in order to display their performance over tests.

6.4.1 Benchmarking

In general, benchmarks can be divided into two types with regard to what they are supposed to evaluate:

Competence Benchmarks allow the level of competence and performance of a particular system to be characterised with regard to a set of well defined tasks. They are helpful for improving individual systems.

Comparison Benchmarks enable systems to be compared with regard to each other on a general purpose task. Their goal is mainly to help in improving the field as whole, rather than individual systems.

6.4.2 Results

The implementation of the current system uses test cases from the OAEI 2007 campaign to evaluate it, as noted above. The experiments were run using a JDK 1.6.0 program on a two processors machine (Intel (R) Core (TM) 2 Duo CPU, 2.4GHz, 3 GB RAM) and Windows. The benchmark dataset was in the domain of bibliography. The OAEI test was run on a group of 52 source and reference ontologies to compare and evaluate the tools. The benchmark data tests were divided into five groups, as shown in Table 6.1.

Table 6.1: Description of Benchmark Data Set

Test Sets	Ontology Description	Num of Ontologies
101-104	Similar in both label description and hierarchy structure	**4**
201-210	Similar hierarchy structure	**10**
221-247	Similar in label description	**18**
248-266	Different in both label description and hierarchy structure	**15**
301-304	Real-world ontologies provided by different institutions	**4**

Therefore, the results are reported below in corresponding groups.

Tests 101-104:

Tests 101, 103 and 104 were straightforward tests of ontology alignment on which our system performed very well, because the ontologies had no special features or difficulties in aligning them. The source ontologies contained concepts and properties with the same names as those in the reference ontologies; therefore, string-, linguistic-, heuristic- and structure-based strategies were employed to find the alignment. However, as the results from the first two strategies applied to the terms in each document were quite different, this combination easily found most of the alignments, whereas the structure-based strategies had little effect on the final results. On the other hand, in test #102 the two candidate ontologies are totally different, therefore, this test created problems for our algorithm; therefore, no alignment was expected for this test, so that precision and recall were automatically set at 0.0 and this value was omitted from the average calculation. In the other three tests, both precision and recall were 1.0. Table 6.2 shows the results for this first group.

Table 6.2: Results of test set 101-104

Test ID	Precision	Recall
101	1.0	1.0
102	0.0	0.0
103	1.0	1.0
104	1.0	1.0
Average (excluding102)	1.0	1.0

Tests 201-210:

The problems with this second test set were that there was no name for some terms, others had changed their names, or there was no comment. These difficulties could not be solved by string-based methods. Nevertheless, our system achieved good results for this test set, so this system is able to apply different strategies in the different tests. Indeed, our system found most correct alignments by using most features of ontologies, such as comments (i.e. content), synonymy, structure and instances. The most important issues are illustrated below and briefly comment on each of these tests.

In the ontology test (201), where labels of concepts and properties were replaced by random ones, there was no similarity between reference and target ontologies in the strategies based on name character strings, so the string-based method was shown not to work. By contrast, the structure-based strategies were successful in producing some matched pairs of concepts and properties. The heuristic-based strategies were utilised to find alignments that could not be found using the strings matcher. In test 202, where neither names nor comments appeared, instance- and structure-matching were used.

These tests were applied to ontologies with no names, which was the case where names had been replaced by random strings, synonyms, naming conventions or even foreign names, but with comments in place (ontologies 201, 204, 205, 206, 207 and

210). These ontologies were found to have low label similarity; on the other hand, they had high structural similarity to the reference ontology.

For example, tests 203 and 208 had high label and name similarity but lacked comments. The use of labels and names, even with conventions, resulted in maximum precision and recall (1.00) for ontology 203 and to high precision (0.98) and recall (0.96) for ontology 208, so they did not affect the performance much. Tests 205 and 209, which shared synonym labels with the reference ontology, were mapped with high precision and good recall.

In fact, through these tests we explicated the comments and utilised the instances, as well as the matching computed by the semantic and structural methods. All these utilisations and matches result in high scores. In fact, throughout these ten tests, precision ranged from 0.84 to 1.0 and recall from 0.76 to 1.0, as Table 6.3 shows.

Table 6.3: Result of test set 201-210

Test ID	Precision	Recall
201	0.94	0.90
202	0.84	0.76
203	1.00	1.00
204	0.98	0.96
205	0.92	0.89
206	0.96	0.96
207	0.97	0.96
208	0.98	0.96
209	0.85	0.78
210	0.95	0.91
Average	0.94	0.91

Tests 221 to 247:

In the third test set, the names, labels and comments had no special features that might confuse the alignment, but the structures of these ontologies were manipulated and some instances or/and properties were added. Therefore, in these ontologies our algorithm performed very well on string-, linguistic- and heuristic-based strategies in computing the similarity between features. This was due to the fact that the terms in these test cases had high string similarity; moreover, the heuristic matcher performed very well in these tests. On the other hand, where specific terms did not have similar names or comments, our algorithm was able to apply structural or semantic features of ontologies in order to derive the remaining alignments.

The most important issues affecting each of these tests were stated above and are briefly restated here. Ontologies 221 to 247 featured no specialisation (221), a flattened hierarchy test (222), an expanded hierarchy test (223), no instances (224), no restrictions (225), no datatypes (226), unit differences (227), no properties (228), class vs. instances (229) and flattened classes (230); all of these were matched with a very high recall and precision rate. As a conclusion, on this group of tests our algorithm performed well, which can be attributed to the fact that this system carried out both syntactic and semantic similarity assessments.

Although the structures of the candidate ontologies were changed, our algorithm found most correct alignments by using strings (label similarity, comment similarity), the linguistic perspective and heuristic matching. Therefore, both precision and recall were excellent.

While tests 221-247 shared the same names and comments, their structures differed. Instances were similar, but some ontologies did not contain them. The information given was sufficient to reach very good results. For most of these tests the structures were modified, which means that structural similarity was low, but the label similarity was very high. Because of this low structural similarity, the structure matcher did not work well for some tests; for example, tests 221, 232, 233 and 241 had high label and structural similarity factors, so both linguistic and structure-based

strategies were employed, although the structure matcher made little contribution. Table 6.4 shows the results.

Table 6.4: Result of tests 221-247

Test ID	Precision	Recall
221	1.00	1.00
222	1.00	1.00
223	1.00	1.00
224	1.00	1.00
225	1.00	1.00
228	1.00	1.00
230	1.00	1.00
231	1.00	1.00
232	1.00	1.00
233	1.00	1.00
236	1.00	1.00
237	1.00	1.00
238	1.00	1.00
239	1.00	0.99
240	1.00	0.99
241	1.00	1.00
246	1.00	1.00
247	1.00	1.00
Average	1.00	0.999

Table 4.6 shows the results which appeared from tests 221-247. These results are very high and are nearly equal to 1, which is the result of our algorithms heavily using linguistic and string matching algorithms.

Tests 248-266:

This next set of benchmark tests was the most difficult. Labels, comments and linguistic characteristics had been heavily changed or replaced by random strings, and the structure changed. In this case, both string (label, comment) and structural similarity between the source and reference ontologies were low, so string similarity was unable to extract meaningful results from this section. Our algorithm performed poorly in some cases, where it was very difficult to identify the correct alignments.

In tests 249, 250 and 257, the structure-based strategies (taxonomy, graph) began to be active to help improve the final alignments. Moreover, the algorithm used lexical matching (i.e. WordNet), the hierarchy and the instances. In fact, the only strategy to contribute to this set of alignments was instance matching. Unfortunately, not all classes and properties had instances, so that information merely helped to find corresponding entities. The results of these tests were therefore reasonable: the recall measure ranged from 0.26 to 0.70, while the precision was far more satisfactory, ranging from 0.81 to 0.93. Table 6.5 shows these results.

Table 6.5: Results of tests 247-266

Test ID	Precision	Recall
248	0.85	0.66
249	0.81	0.70
250	0.88	0.55
251	0.87	0.59
252	0.85	0.65
253	0.88	0.68
254	0.89	0.26
257	0.83	0.49
258	0.84	0.56
259	0.86	0.59
260	0.83	0.39
361	0.84	0.29

262	0.93	0.25
265	0.83	0.39
266	0.84	0.27
Average	0.86	0.49

Tests 301-304:

Tests 301-304 were applied to real-world ontologies which had been modelled by different institutions but for the same domain of bibliographic metadata. For these tests, which combined the difficulties of all previous tests, the whole system was utilised in order to obtain good results. Overall, precision ranged from 0.83 to 0.95 and recall from 0.70 to 0.94.

In tests 302 and 303 the names (labels) were often dissimilar, there were no comments and the structures were quite different. Therefore, our system used auxiliary information in order to find alignment. The only test of this set to have instances was 304; this lack of instances and of semantic matching was a key cause of the low recall measure, but this was still reasonable. The ability of the linguistic-based strategies to find similarities between the two candidate ontologies in each test was high, while the performance of the structural strategies was reasonable. Finally, the heuristic-based strategies played a key part in improving the matching results, shown in Table 6.6.

Table 6.6: Results of tests 301-304

Test ID	Precision	Recall
301	0.89	0.81
302	0.83	0.72
303	0.85	0.79
304	0.95	0.94
Average	0.88	0.82

Table 6.7: Full results of all benchmark tests

Test ID	Precision	Recall	F-measure
101	1	1	1
103	1	1	1
104	1	1	1
201	0.94	0.9	0.92
202	0.84	0.76	0.8
203	1	1	1
204	0.98	0.96	0.97
205	0.92	0.89	0.9
206	0.96	0.96	0.96
207	0.97	0.96	0.96
208	0.98	0.96	0.97
209	0.85	0.78	0.81
210	0.95	0.91	0.93
221	1	1	1
222	1	1	1
223	1	1	1
224	1	1	1
225	1	1	1
228	1	1	1
230	1	1	1
231	1	1	1
232	1	1	1
233	1	1	1
236	1	1	1
237	1	1	1
238	1	1	1
239	1	0.99	0.99
240	1	0.99	0.99
241	1	1	1
246	1	1	1
247	1	1	1
248	0.85	0.66	0.74
249	0.81	0.7	0.75
250	0.88	0.55	0.68
251	0.87	0.59	0.7
252	0.85	0.65	0.74
253	0.88	0.68	0.77
254	0.89	0.26	0.4
257	0.83	0.49	0.62
258	0.84	0.56	0.67
259	0.86	0.59	0.7
260	0.83	0.39	0.53
361	0.84	0.29	0.43
262	0.93	0.25	0.4
265	0.83	0.39	0.53
266	0.84	0.27	0.4
301	0.89	0.81	0.85
302	0.83	0.72	0.77
303	0.85	0.79	0.82
304	0.95	0.94	0.94
Average	0.935	0.813	0.87

Table 6.7 shows the overall results obtained in the OAEI2007 tests. These indicate that our system was able to provide good results in both precision and recall. In fact, the precision measure is more important for all systems, because it gives us the chance to illustrate constructive conclusions as to how the handling of uncertainty can influence the precision of a system. On the other hand, the recall measure can be used effectively to identify where further improvements are needed.

In order to aggregate the results of all the tests into groups and find the average of each group, Table 6.8 shows the average performance of each group and the overall performance on the benchmark test set. These results are shown graphically in Figures 6.1-6.3.

Table 6.8: Comparative performance of the algorithm in different test sets

Tests / Measure	101-104	201-210	221-247	248-266	301-304
Precision	**100**	**94**	**100**	**86**	**88**
Recall	**100**	**91**	**99.9**	**48.8**	**82**
F-measure	**100**	**92**	**99.99**	**62**	**84.6**

Figure 6.1 shows the precision of our system for all tests, indicating that the system worked very well, as most results were very good, in the range 86% -100%.

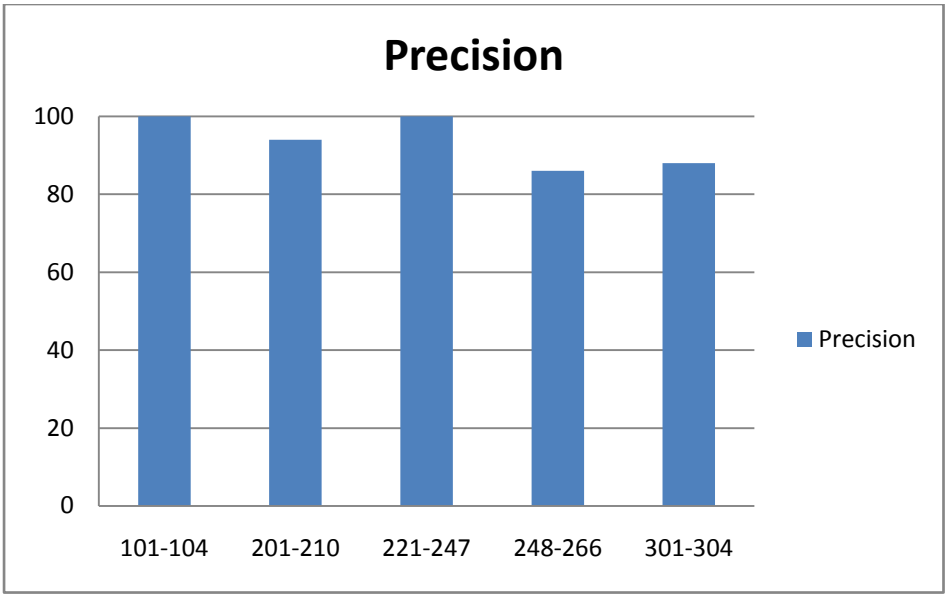


Figure 6.1: Precision results for all tests

Figure 6.2 shows that the recall results, in the range 49% -100%, were mostly very good, but very poor in tests 248-266, where there were many changes between the candidate and reference ontologies. By contrast, our system provided perfect results in tests 101-104 and 221-247, while the results achieved in tests 201-210 were very good and those in the real-world tests (301-304) were reasonably good.

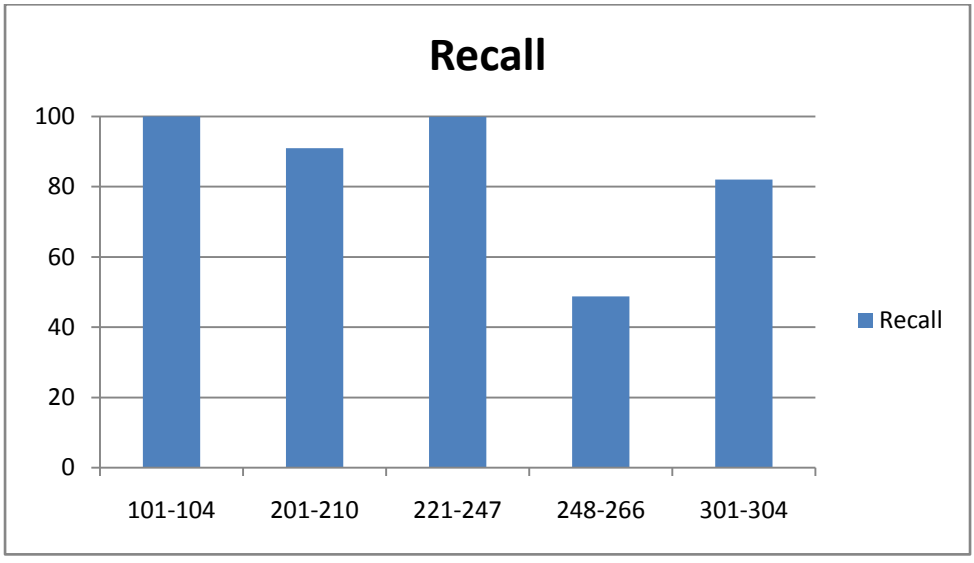


Figure 6.2: Recall results of all tests

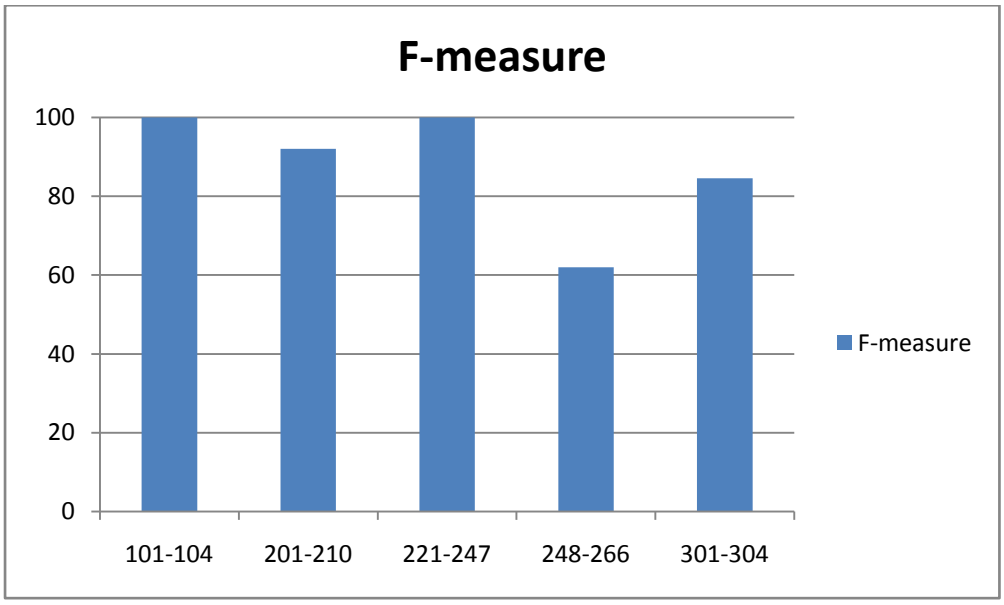


Figure 6.3: F-measure results for all tests

Finally, Figure 6.3 shows the F-measure results for all tests. As this indicates the balance between precision and recall, the results were very good except for tests 248-266, for reasons explained above.

6.5 Comparison with other existing approaches

In order to evaluate our system, a comparison of the system results was made against the published results from the 2007 Ontology Alignment Evaluation Initiative. The algorithm that performed best in the 2007 contest was ASMOV [44]. Therefore, our system is compared with ASMOV [44], Lily [94], RiMOM [55], Falcon [36], OLA2 [30], DSSim [72], Prior+ [59] SEMA [85] and X-SOM [17]. In fact, there were more systems in this test, but they were weak and did not perform in all tests, so they are omitted from this comparison. The first four were the top ranking algorithms in terms of ontology alignment and provided very good results compared with OAEI 2007 competitors. See table 6.9 for more result.

Table 6.9 Comparison of Experimental Results (Precision & Recall of OAEI 2007 Competitors)

algo	STRL		X-SOM		ASMOV		DSSim		FALCON		LILY		OLA		PRIOR+		RiMOM		SEMA	
test	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.
101	1	1	1	0.98	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
103	1	1	1	0.98	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
104	1	1	0.97	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
201	0.94	0.9	0.81	0.81	1	1	1	0.16	1	0.95	1	1	0.85	0.85	0.96	0.94	1	1	0.92	0.98
202	0.84	0.76	0.82	0.82	0.88	0.88	1	0.16	0.87	0.87	1	0.8	0.84	0.84	0.87	0.63	1	0.8	0.74	0.3
203	1	1	n/a	n/a	1	1	1	1	1	1	1	1	1	1	1	1	1	0.88	1	1
204	0.98	0.96	0.99	0.69	1	1	0.96	0.91	0.98	0.98	1	1	1	1	1	1	1	1	0.95	0.96
205	0.92	0.89	0.72	0.71	1	1	0.94	0.33	1	0.98	1	0.99	0.92	0.92	0.97	0.95	1	0.99	0.93	0.96
206	0.96	0.96	0.74	0.73	1	0.99	0.97	0.39	1	0.93	1	0.99	0.99	0.99	0.98	0.96	1	0.99	0.94	0.97
207	0.97	0.96	0.69	0.68	1	0.99	0.97	0.39	0.98	0.91	1	0.99	0.97	0.97	0.98	0.96	1	0.99	0.92	0.97
208	0.98	0.96	0.99	0.75	1	1	0.95	0.9	1	1	1	1	1	1	1	0.95	0.98	0.86	0.89	0.8
209	0.85	0.78	0.7	0.69	0.92	0.9	0.91	0.32	0.79	0.78	0.92	0.91	0.7	0.69	0.83	0.59	1	0.84	0.82	0.61
210	0.95	0.91	0.7	0.69	0.97	0.95	0.97	0.39	0.81	0.8	1	0.91	0.87	0.87	0.97	0.79	0.99	0.85	0.81	0.47
221	1	1	1	0.99	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
222	1	1	1	0.98	1	1	1	1	1	1	1	1	0.99	1	1	1	1	1	0.96	1
223	1	1	1	0.98	1	1	1	1	1	1	1	1	0.99	1	1	1	1	1	0.97	0.98
224	1	1	1	0.98	1	1	1	1	1	0.99	1	1	1	1	1	1	1	0.99	1	1
225	1	1	1	0.98	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
228	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
230	1	1	0.99	0.97	0.99	1	0.97	1	0.94	1	0.94	1	0.92	1	0.94	1	0.94	1	0.75	1
231	1	1	n/a	n/a	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
232	1	1	1	0.99	1	1	1	1	1	0.99	1	1	0.99	1	1	1	1	0.99	1	1
233	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
236	1	1	0.97	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
237	1	1	1	0.98	1	1	1	1	1	0.99	1	1	0.98	1	0.99	1	1	0.99	0.96	1
238	1	1	1	0.99	1	1	1	1	1	0.99	0.98	0.98	1	1	1	1	1	0.99	0.97	0.97
239	1	0.99	0.97	1	0.97	1	0.97	1	1	1	0.97	1	0.94	1	0.97	1	1	1	0.94	1
240	1	0.99	0.97	1	0.97	1	0.97	1	1	1	0.97	1	0.94	1	0.97	1	1	1	1	1
241	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
246	1	1	0.97	1	0.97	1	0.97	1	1	1	0.97	1	0.94	1	0.97	1	1	1	0.94	1
247	1	1	0.97	1	0.94	0.97	0.97	1	1	1	0.94	0.97	0.94	1	0.97	1	1	1	0.94	0.94
248	0.85	0.66	0.75	0.75	0.86	0.82	1	0.16	0.85	0.84	1	0.77	0.84	0.78	0.76	0.45	0.99	0.78	0.73	0.3
249	0.81	0.7	0.6	0.6	0.89	0.89	1	0.16	0.87	0.87	1	0.8	0.79	0.79	0.88	0.63	1	0.79	0.73	0.28
250	0.88	0.55	0.18	0.18	0.91	0.3	1	0.27	1	0.27	0.85	0.67	0.75	0.27	0.68	0.58	1	0.55	1	0.27
251	0.87	0.59	0.45	0.45	0.83	0.77	1	0.17	0.56	0.56	0.96	0.74	0.73	0.74	0.71	0.4	0.76	0.58	0.65	0.26
252	0.85	0.65	0.49	0.49	0.87	0.87	1	0.16	0.71	0.71	0.94	0.76	0.79	0.79	0.69	0.39	0.85	0.7	0.65	0.25
253	0.88	0.68	0.54	0.54	0.85	0.81	1	0.16	0.85	0.84	0.97	0.75	0.79	0.74	0.76	0.45	0.99	0.77	0.71	0.28
254	0.89	0.26	0.03	0.03	0.83	0.3	1	0.27	1	0.27	1	0.27	0.9	0.27	1	0.27	1	0.27	1	0.27
257	0.83	0.49	0.12	0.12	0.91	0.3	1	0.27	1	0.27	0.85	0.67	0.75	0.27	0.69	0.61	1	0.55	1	0.27
258	0.84	0.56	0.32	0.32	0.82	0.76	1	0.17	0.54	0.54	0.76	0.74	0.74	0.75	0.71	0.39	0.76	0.57	0.66	0.25
259	0.86	0.59	0.33	0.33	0.87	0.87	1	0.16	0.7	0.7	0.94	0.75	0.8	0.8	0.73	0.37	0.85	0.69	0.68	0.26
260	0.83	0.39	0.03	0.03	0.78	0.24	0.9	0.31	1	0.31	0.62	0.45	0.77	0.34	0.67	0.34	0.93	0.45	1	0.31
261	0.84	0.29	0.03	0.03	0.91	0.3	0.9	0.27	0.89	0.24	0.61	0.42	0.69	0.27	0.45	0.39	1	0.27	1	0.27
262	0.93	0.25	0	0	0.83	0.3	1	0.27	1	0.27	1	0.27	0.9	0.27	1	0.27	1	0.27	1	0.27
265	0.83	0.39	0	0	0.77	0.34	0.9	0.31	1	0.31	0.86	0.41	0.69	0.31	0.67	0.34	0.93	0.45	1	0.31
266	0.84	0.27	0.03	0.03	0.91	0.3	0.8	0.24	0.89	0.24	0.64	0.42	0.69	0.27	0.45	0.39	1	0.27	1	0.27
301	0.89	0.81	0.91	0.49	0.93	0.82	0.82	0.3	0.91	0.82	0.89	0.8	0.71	0.66	0.93	0.82	0.75	0.67	0.7	0.75
302	0.83	0.72	1	0.58	0.68	0.58	0.85	0.6	0.9	0.58	0.82	0.65	0.51	0.5	0.82	0.67	0.72	0.65	0.62	0.6
303	0.85	0.79	0.9	0.73	0.75	0.86	0.85	0.8	0.77	0.76	0.58	0.69	0.41	0.82	0.81	0.8	0.45	0.86	0.55	0.8
304	0.95	0.94	0.96	0.87	0.95	0.96	0.96	0.92	0.96	0.93	0.91	0.97	0.89	0.97	0.9	0.97	0.9	0.97	0.77	0.93
Ave	0.935	0.813	0.722	0.686	0.935	0.84	0.97	0.638	0.935	0.81	0.938	0.851	0.878	0.815	0.8936	0.786	0.9558	0.825	0.896	0.722

Table 6.9 show the results for all systems over each individual test and the overall average.

The results for the systems on each of the data sets are shown in Table 6.10 and Figure 6.4. The results of STRL are compared with those obtained from all other existing algorithms which were used in the 2007 Ontology Alignment Evaluation Initiative; the details are given in Table 6.10.

Table 6.10: Comparison of Experimental Results (F-measure of OAEI 2007 Competitors)

Test	OLA2	ASMOV	Lily	DSSim	Prior+	Falcon	RiMOM	SEMA	X-SOM	STRL
101	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**99**	**100**
103,104	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**98.75**	**100**
201-210	**90.8**	**97.4**	**97.5**	**60.4**	**91.6**	**84.3**	**95.8**	**85.5**	**78.5**	**92**
221-247	**98.8**	**99.4**	**99.2**	**99.6**	**99.4**	**99.7**	**99.7**	**97.5**	**99**	**99.9**
248-266	**64**	**76.5**	**72.9**	**36**	**57**	**56.4**	**73.4**	**51**	**26**	**62**
301-304	**68**	**83.5**	**78.8**	**72.6**	**84**	**82**	**74.6**	**73**	**80.5**	**84.6**
Average	**86.9**	**92.8**	**91.4**	**78.1**	**88.6**	**87**	**90.6**	**84.5**	**80.3**	**89.7**

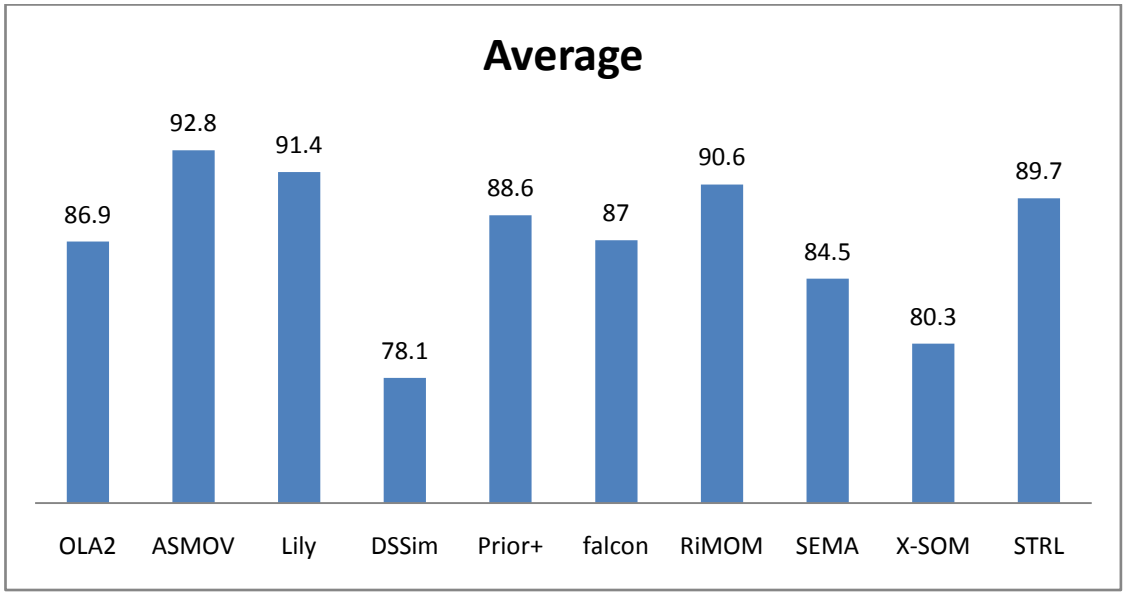


Figure 6.4: Average F-measure for systems on OAEI 2007 benchmark tests

Figure 6.4 shows the average F-measure for ten systems include ours (STRL) on OAEI 2007 benchmark tests. Based on the benchmark data set, STRL considerably outperforms X-SOM and SEMA and is slightly ahead of OLA2, Prior+ and Falcon, but ASMOV, Lily and RiMOM are slightly ahead of STRL. Thus the STRL system performed fourth best overall.

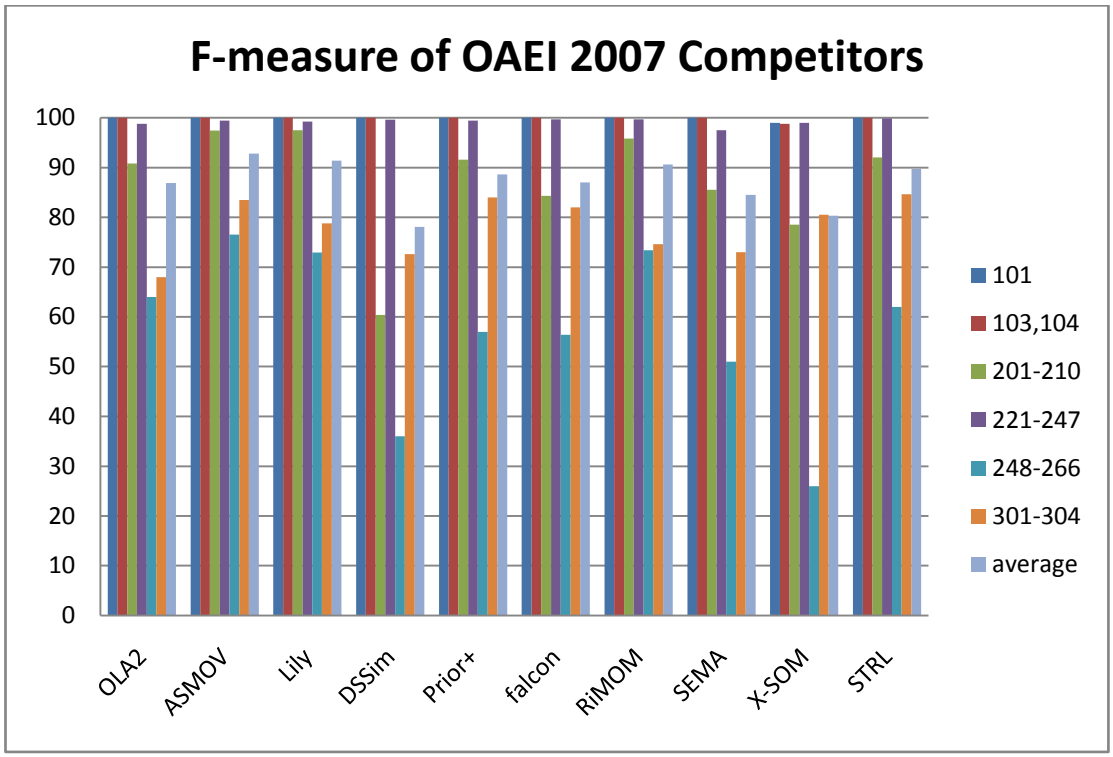


Figure 6.5: F-measure of all test groups for all systems of the OAEI 2007

The observations to be made from Table 6.9 and Figure 6.5 are as follows. First, tests 101,103,104 were basic ontology mapping tests. In fact, the source ontologies contained concepts and properties with the same names as those in the reference ontologies and had comments and instances; therefore, there were no special difficulties in aligning them and the results were perfect, with precision and recall at 1.0. Thus, it was impossible to distinguish our system, because nine of the ten systems gave the same result; on the other hand, this gave a good indication of the

performance of our system, showing that linguistic-based strategies can easily find most of the alignments.

In fact, for the first group of tests our system performed very well. The strong performance in this group reflects the fact that the basic ontology entities remained unchanged with respect to the reference ontology.

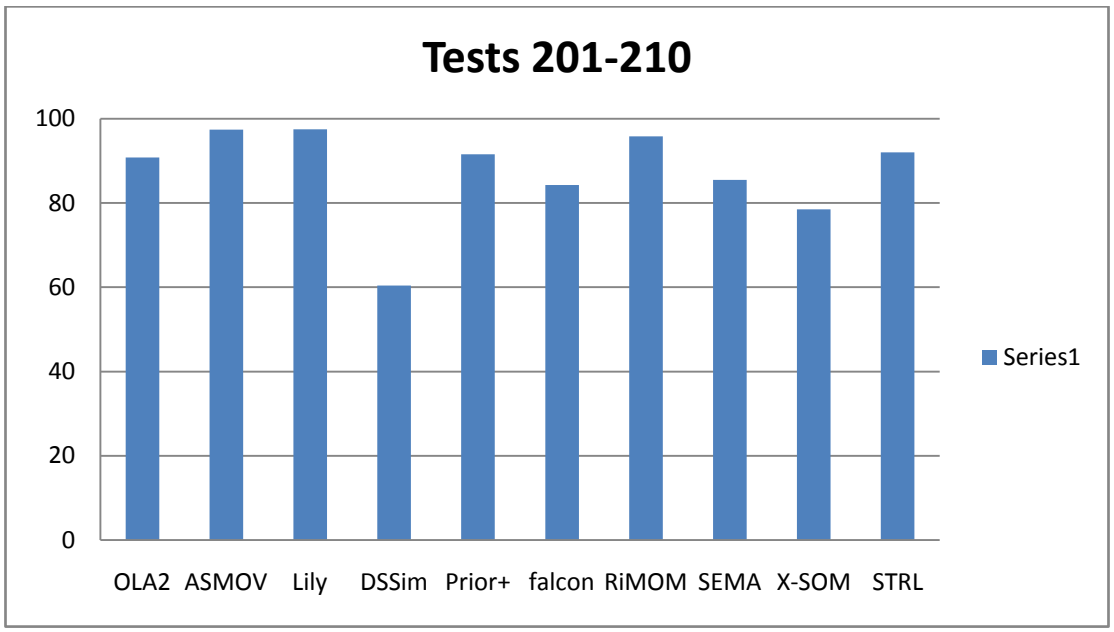


Figure 6.6: Results of tests 201-210 for all systems

As for the next set (201-210), many changes were made on these tests, making it difficult use linguistic strategies. Indeed, the reference and test ontologies were similar in their structures but different linguistically. Therefore, our system used all available features of ontology to achieve high precision and recall. In this group of tests our system attempted to match names and comments when they were available, as in tests 203, 204 and 208, while it also sought to match instances and comment in e.g. tests 201, 206, 207 and 210. On the other hand, in cases where most of the linguistic information was missing, our system used structure matching in order to obtain useful results, as in tests 202 and 210. Finally, this system applied external

resources (a lexical dictionary) with tests 209 and 205, as the test ontology contained many synonyms.

Our system performed very well (F-measure = 92), but less well that ASMOV, Lily and RiMOM, which achieved 97.4, 97.5 and 95.8 respectively, putting STRL in fourth position for this group of tests, above seven other systems.

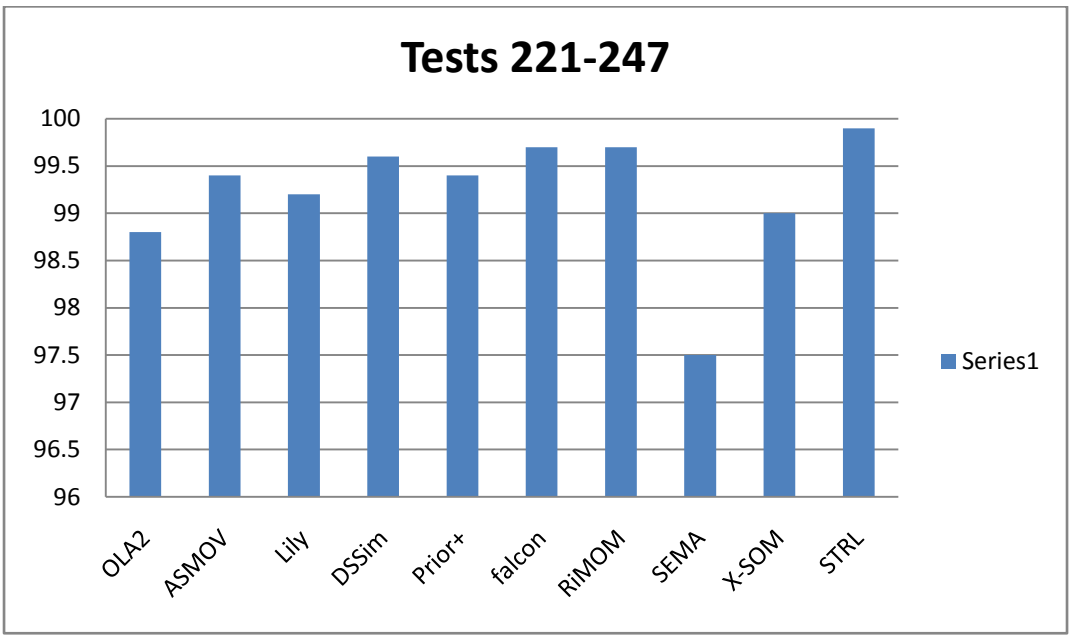


Figure 6.7: Results of tests 221-247 for all systems

For most of tests 221-247, the structures were manipulated, so that structural similarity was low; however, names, labels and comments in these ontologies had no special features, so linguistic similarity was very high. The information given was sufficient to yield very good results. In this set of tests, where the ontologies had high similarity with the reference ontology on linguistic information, our system performed very well and was the best, with precision, recall and F-measure scores of 1.00, 0.999 and 0.999 respectively. Other systems, including Falcon, DSSim and RiMOM also performed very well, with results on the F-measure of 0.997, 0.996 and 0.997 respectively.

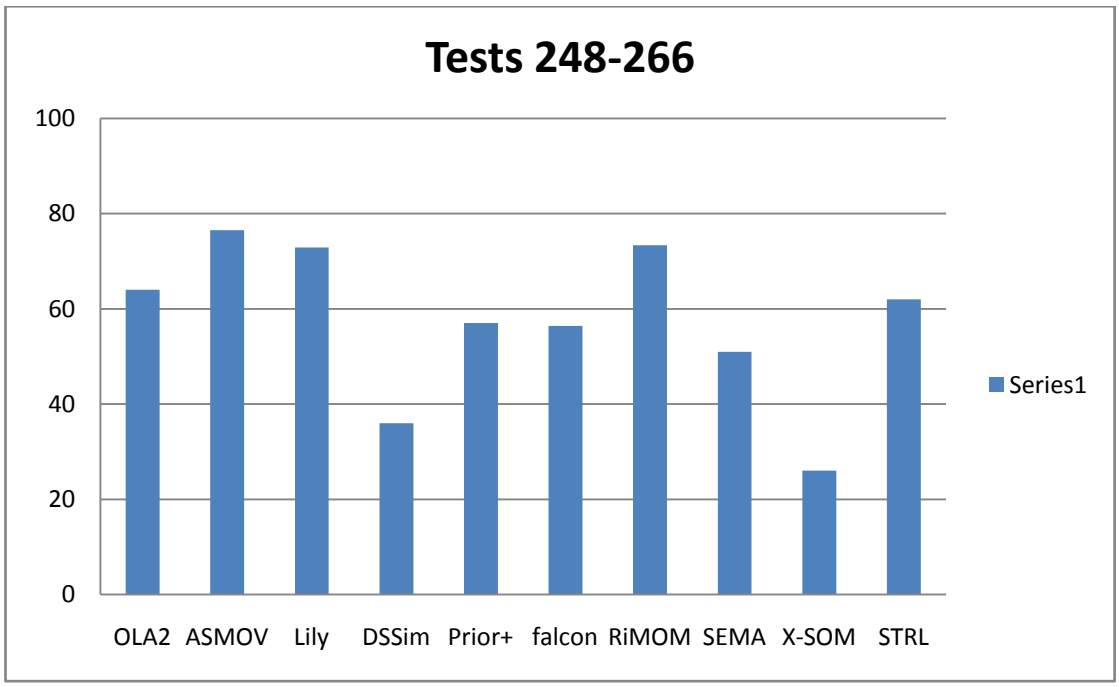


Figure 6.8: Results of tests 248-266 for all systems

In tests 248-266 there were no names and no comments, so linguistic features are totally unavailable. In fact, this group had very low similarity from both linguistic and structural perspectives. Consequently, the only features available were structural and instance information; while the structure feature was relatively weak, the instance information enriched the process by integrating all descriptive information on both classes and properties. Therefore, our system achieved reasonable but not good results, placing STRL in fifth position after OLA2, ASMOV, Lily and RiMOM on the F-measure, for which the scores were 64, 76.5, 72.9 and 73.4 respectively, while that for STRL was 62.

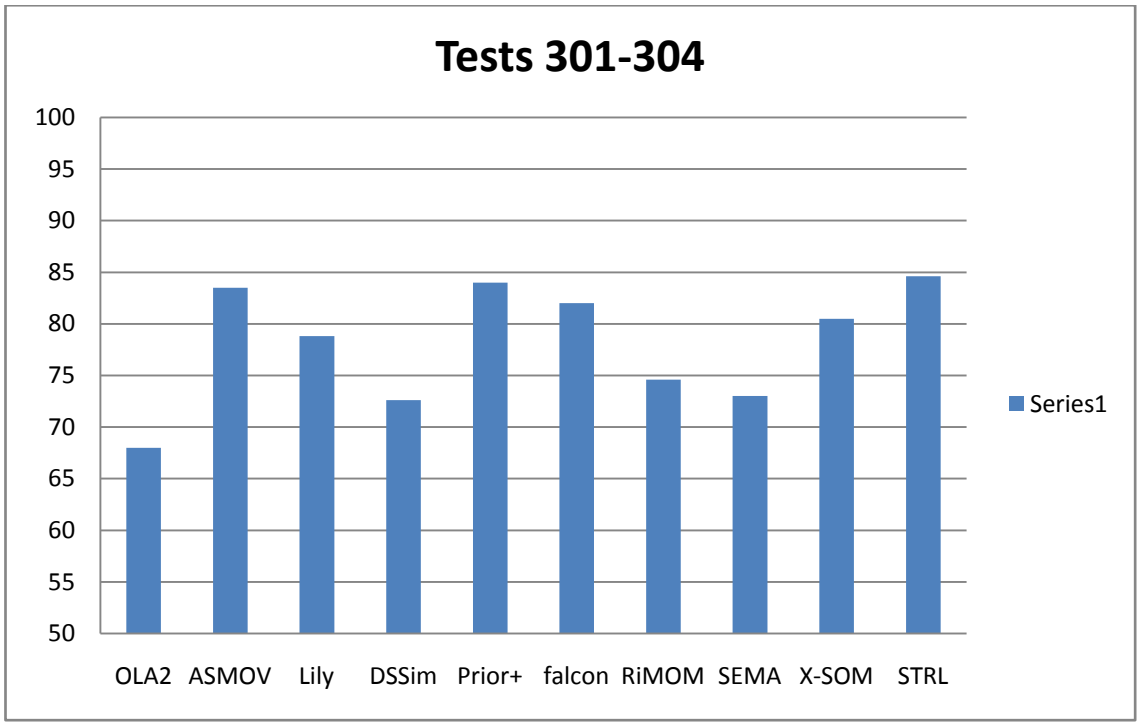


Figure 6.9: Results of tests 301-304 for all systems

The real-world tests (301-304) were conducted on ontologies created by different institutions but for the same domain of bibliographic metadata; consequently, their linguistic similarity was high but their structural similarity was low. Our system considered the labels in the ontologies to be a very important feature, with additional linguistic information like comments and instances. Moreover, the STRL utilised an auxiliary thesaurus to explore synonymous relations between concepts in ontologies. The results show that our system achieved 84.6%, making STRL the best system on the real-world cases. Some other systems, such as ASMOV, Prior+, Falcon and X-SOM also achieved good results (83.5, 84, 82 and 80.5% respectively).

6.6. Case Study: Interoperability

In agent communication, ontology can be used in order to describe the meaning of concepts, while in a multi-agent system an ontology is considered the basis for communication. From another point of view, ontologies are considered to provide the solution to data heterogeneity on the web. Several ontologies have already been developed and made publicly available, and several applications are needed to access or use them. However, the existing ontologies themselves can be heterogeneous. In fact, a single ontology is no longer enough to support all tasks. Hence, alignment between these ontologies has become necessary, to provide a common layer by which information can be exchanged in a semantically sound manner.

In order to deal with this problem we have to establish semantic interoperability which depends upon finding relationships between terms which belonging to different ontologies. We call this process "ontology alignment". In fact, semantic alignment between ontologies is a necessary precondition to discover the alignment across ontologies or to establish interoperability between agents or services using different ontologies.

Structural Heterogeneity is a problem which occurs when the same information is represented in different ways in different ontologies, or even when information within the same ontology is represented in different ways. This is considered a problem for heterogeneous databases but not for agents. In order to deal with it, two types of matching were maintained based on the structure of ontologies: in one case on taxonomy and in the other on semantic structure. In semantic structure matching, a terminological matcher was combined with the internal structure in order to deal with problems at this level.

In order to establish interoperability between agents or services using different ontologies, semantic alignment between ontologies became a necessary precondition.

Semantic Interoperability means ensuring that the precise meaning of the information relayed by one system can be understood by other systems and

applications, even if they are not tailored for this specific information exchange. In other words, it emphasises the importance and usefulness of information sharing, enabling information from different systems to be combined with other information resources in a meaningful way. The achievement of full interoperability between independent systems requires technical, organisational and semantic interoperability. Since technical interoperability now works even better, the problems that may arise when important information is interpreted differently come increasingly into focus. Misunderstanding frequently causes more or less serious problems and in worst cases can lead to wars and disasters. The need for better semantic interoperability has always existed and as such is technology neutral.

Semantic interoperability is used to solve the conflicts that occur in semantics among heterogeneous information sources. Semantic conflict occurs whenever two systems do not use the same information. Format conflict in the simplest of information systems may take the form of homonymy (use of the same words that have different meanings) and synonymy (use of different words with the same meaning).

The calculation of semantic similarity between concepts is the first step in creating agreement between the ontology and the content / service provider or the content of the request for information. An individual concept can be represented as a hierarchy according to the label that contains some of the information or semantic structure.

In interoperability or integration processes in general, there are several ways to overcome semantic differences by utilising the concept of ontology. The first model is to create ontology that is understood by all sources of information. This global approach, which is suitable for a static source of information, is often termed *ontology merging.* In the second model, each source of information has its respective ontology and interoperability is created through a process of alignment of one ontology to the other.

Semantic heterogeneity concerns the intended meaning of information. Therefore, we have distinguished the differences between ontologies as follows:

1. Different semantic structures (so-called structural conflict). This problem was solved by semantic matching and a heuristic matcher, which was based on combining the internal structure of the ontology with a terminological matcher.

2. Different names (so-called naming conflicts). This problem appears when the same name or information represents different information. This problem was solved by using a multi-matcher:

   - SoftTF/IDF is applied to comments on each class or property in order to find similarities based on text explaining the nodes in natural language.

   - Soundex is applied to solve problems arising from the use of some natural language (slang) comments.

   - Linguistic matching is based on thesauri in order to solve homonym and synonym problems that could arise from using natural language.

   - The heuristic matcher took many features of ontologies and combined them in order to compute the similarity between elements iteratively.

3. Different representations of the same data. This kind of conflicts occurs when different formats, units or expressions have been used for the same data (so-called data conflicts). This problem was solved by using:

   - Linguistic matching based on thesauri; and

   - Jaccard similarity, a measurement of similarity based on instances of classes.

6.7 Summary

The core aim of this work was to develop new ontology alignment technique by using different matching strategies. This chapter has reported work to evaluate our system, check its performance and compare it with other well known systems. This new ontology alignment approach, which is called STRL, utilises both linguistic and structural information from ontologies in order to solve ontology alignment problems.

The results discussed here have been calculated with four matchers, whose performance concerns four individual similarities, i.e. string-based, linguistic-based, heuristic-based and structure-based. For the combination of the match results the average value has been computed and a selection has been made, using e.g. a threshold.

- String- and linguistic-based similarity are intuitive and work very well when test ontologies are highly linguistic and similar to the reference ontology (i.e. containing labels, comments, instances, synonyms, datatypes and properties).

- Structure-based similarity (using taxonomy and semantic matchers) contributed well to most tests, especially in cases where there was very little or no linguistic information.

- Heuristic-based similarity depends on both linguistic and structural information from ontologies and thus its overall performance is superior to any of the individual similarities.

The benchmark tests have provided very valuable evidence on how the alignment approach behaves. These organised tests constitute a high-quality fundamental test base. In order to check the performance of this system, the evaluation criteria used by the OAEI ontology matching campaign 2007 is followed, comparing our system with the best performing algorithms that participated in the benchmark evaluations. This participation has been shown to be rather valuable for improving any tool; many improvements and other methodological changes were made in order to be able to deliver the above precision and recall results in the OAEI 2007 tests.

As many others did, the benchmark tests were divided into five groups in order to assess the performance of this system and to compare it with others. This comparison showed that:

- On tests 101-104 all algorithms, including ours, created the alignment with high precision.

- On tests 201-210 our system performed very well, finding most correct alignments by using most of the features of the ontologies, such as comments, instances and synonyms, but not as well as Lily, ASMOV or RiMOM.

- On two other groups of tests (101-104 and 221-247), we found that the linguistic information in the ontologies was very similar to that of the reference ontology. On the other hand, there was much less intervention, such as randomly generated names of classes/properties. Therefore, it was easy to find useful features like string matchers or linguistic matchers that could contribute to the matching.

- On tests 221-247 our system was the best, finding all correct alignments by using the information from labels and comments. On the other hand, most systems performed well.

- Tests 248-266 were the most difficult set. Our system performed quite well.

- Tests 301-304 were on real ontologies; again, our system provided the best performance.

In fact, our system (STRL) performed very well on all benchmark tests. Remarkably, it outperformed all others on benchmark tests 221-247 and 301-304.

Chapter 7 Conclusion and Future Work

7.1 Conclusions

The ultimate goal of my research was to solve the problem of ontology alignment, thus enabling semantic interoperability between different applications. This thesis introduces a framework based on ontologies and their alignment. More specifically, a new generic approach has been developed to the semi-automatic matching of ontologies with minimum human effort. Therefore, a main contribution of this thesis is to present a novel stepwise process with multi-matching strategies that can significantly improve the ontology alignment process itself and its output.

In the Semantic Web, the area of particular concern to this study, data will be extracted from many different ontological structures, and information processing across ontologies is not possible without knowledge of the semantic mappings between them. Therefore, semantic alignment between ontologies is a necessary precondition for discovering the alignment across ontologies or establishing interoperability between agents or services using different ontologies.

In the field of ontology matching, one of the main issues is the need for flexible algorithms and tools, capable of adapting to different domains and also to different interpretations of the notions of alignment and similarity. Our system implements a variety of techniques that are based on terminology, which is itself based on the study of words in conceptual or structural labels, which are in turn based on the relative positions of concepts in the taxonomies.

Our approach brings together techniques in modelling, string matching, linguistics, structure matching and heuristic matching in order to provide a semi-automatic alignment framework for the purpose of improving semantic interoperability in heterogeneous systems. Such ontology alignment means linking entities of source ontology with those of target ontology based on different features of these ontologies and using different strategies. Thus, ontologies are fed into the ontology alignment

system and a set of alignment elements is returned. For the end-user, this work will provide an easy-to-use tool for ontology alignment.

Our system compares many features of ontology in order to find similarities: it compares the labels of entities, the relations among them and their known extensions (instances and classes), as well as their internal structure (e.g. the value range or cardinality of their attributes). The main objective of these measures is to automate the process of comparing alignment methods and the assessment of the quality of their products. The objective of the work is to introduce a method for finding semantic correspondences among ontologies over different applications in the same domain, in order to improve interoperability across heterogeneous systems.

Some approaches or tools are based on a single map matching strategy, while others presuppose more than one strategy; these are called multi-strategy or hybrid matchers. It seems that a multi-strategy algorithm is actually better than a single-strategy one, because it deals with and solves more than one critical problem. It can be said that a system or tool which has more than one matching strategy is likely to be more readily applied in different domains.

In fact, most systems or tools use multiple matching algorithms to cope with different types of mismatch between ontologies; selecting the matching algorithms depends on the application domain.

Indeed, all criteria for success are met throughout this thesis by following our algorithm until reaching the results which are presented on Chapter 6.

There are two levels of ontology-matching techniques:

- Element-level matching techniques compute mapping elements by analysing entities without reference to their relations with other entities. This level may be language-based and can include string-based techniques that are widely used in matching systems, such as prefix, suffix and n-gram (used to compute the number of common n-grams for two strings). Distance measures are applied in the semantic matching approach to determine correspondences

between entities, terms or classes in different ontologies, such as equivalence, more general, subsumption, disjointedness/ incompatibility and overlapping. On the other hand, confidence measures are computed to express the equivalence relation between entities that are usually given values in the [0, 1] range. The level could also be linguistic resources-based, considered useful when dealing with natural language processing (used in auxiliary common knowledge thesauri such as WordNet), or constraint-based, which copes with the internal constraints being applied to the definitions of entities and their relationships, such as types and attributes.

- In structure-level techniques, the computing elements are mapped by analysing how entities appear together in a structure. This level could contain graph-based techniques, techniques based on repositories of structures and model-based techniques. These matching approaches use confidence measures. Moreover, structures such as graphs and labels are used as semantic maps; this structural aspect provides help in case the labels are not sufficiently expressive.

7.1.1 Highlights of System

Pre-processing (Feature Generation)

As a first step, the system starts by loading two ontologies and extracting useful ontological features, such as class names and properties. Then normalisation was carried out on these elements by removing stop words, for example.

Alignment (Group of Matchers)

The system explores four different measures of similarity based on strings, linguistics, heuristics and structure, to support ontology alignment. In general, the similarity between entities needs to be calculated in order to find the correspondence between ontological entities. This thesis has described different strategies (e.g. string similarity, synonyms, structural similarity and a strategy based on instances) for achieving similarity between entities.

- **String-based Strategies:** well-known algorithms are used to compute an optimal one-to-one matching in order to assess the similarity between two classes as a function of the similarities between the names of their properties. Our system considers the labels in the ontologies to be very important features, while any additional information, such as comments or linguistic elements, is also very important. Comments in ontologies were shown to work very well when the ontologies were created in a well controlled academic environment.

- **Linguistic-based Strategies:** The WordNet dictionary was used to calculate similarities between words for the comparison of node names in ontologies. Combining numerous feature-matching approaches leads to radically higher quality matching. Therefore, in this part, the string matchers with thesauri are used in order to identify synonyms and hypernyms.

- **Structure-based Matcher:** The results from this matcher were calculated from a detailed analysis of graphs, using many features of ontology such as taxonomy (SuperClass and SubClass relations). Many rules were also applied to the graphs.

- **Heuristic-based Matcher:** This algorithm tries to find matches by analysing the structural similarity between ontologies and can be combined with label similarity measures to produce matching correspondences. The most important feature of ontology is labels, which alone were shown to be capable of returning very satisfactory results.

Post Processing:

This allows the user to choose the ontologies to be aligned and to accept, delete or add matching entities to the final set of alignments.

- Similarity Aggregator.

- Similarity Evaluator.

7.2 Limitations

The limitations of the research are as follows.

- This study was concerned with matching only pairs of ontologies from the same or a similar domain. This restriction is based on the idea that in the real world, the possibility of interaction between two information systems from completely different domains is quite small.

- This research did not involve working across languages; the ontologies to be aligned used English only. However, this system works with a number of description languages (e.g. OWL-Lite, OWL-DL, and RDF).

- For this study, the system is concerned with and restricted to one-to-one or one-to-many matching cardinality, owing to the limitations of the matching extraction algorithm.

7.3 Main Contributions

- An in–depth review to the definition and structure of ontology. Also, various operations over ontologies and a different set of ontology matching methods have been proposed

- A detailed review state-of-the-art of the ontology languages which are used to express ontology over the Web; all these terms were shown in order to provide a basic understanding of ontologies and of description logics, which are the basis of ontology languages.

- A comprehensive review of existing ontology alignment tools. Several existing ontology mapping methods were analysed and compared, since before creating a new approach it is essential to fully understand related work. Here, this means both theoretical work and existing approaches to the alignment of ontologies or other well-defined structures, including their weaknesses, which must be understood if they are to be improved.

- Exploring multiple similarities, such as edit-distance-based similarity, profile similarity and structural similarity, to support ontology mapping.

- Methodology for designing and developing an ontology alignment approach to find the semantic correspondences between elements in different ontologies. Also, a method for semantic enrichment and discovery of semantic correspondences between ontologies has been proposed, which contributed to the understanding of semantic distance between ontologies in general.

- Improving a semantic matcher based on combining lexical matcher with several rules and facts.

- Developing a new heuristic matcher which provided very high quality results.

- Developing techniques that were shown to work over several domains and to provide correct results by using most features of ontologies.

- Multiple matching techniques were combined in order to obtain high quality results; these techniques were string-, structure-, linguistic- and heuristic-based. In fact, different matching methods have been adapted to deal with different sets of features and to cope with ontology mismatching.

- Implementing all components of our system. Also, illustrating examples of each component. To this end, a very efficient calculation of alignments can be achieved by using most features of ontology in order to select the most promising alignment candidates for comparison. This allows the reuse of one ontology alignment approach for many scenarios.

- Providing an analysis of the implementation and evaluation of the method in empirical experiments, presenting the results of the validation experiments that evaluated our system against top ranking competitors.

- Achieving high quality results with efficiency throughout applying the system to real-world scenarios. Also, achieving high scores in terms of both accuracy

and completeness when applying the system to ontologies rich with linguistic information.

- Illustrating the application of our system to a case study (a multi-agent system) and showing its ability to deal with this case.

7.4 Future Work

While the system discovered most of the matching results and ranked them in a useful manner, it is still relevant to ask what prevents the system from achieving the best precision and recall figures. Therefore, based on the results, the study has been identified the following possibilities for improvement to our system:

- Given our findings, there is still work to be done. For example, to extend one-to-one or one-to-many mapping to many-to-many or many-to-one mappings, so the system requires improving the mapping extraction algorithm and adjusts constraints.

- The system requires improving the matching ability, based on more advanced structure matching and the advanced semantic matching methods that have been used in other lines of related research. There is a need to integrate an external repository to analyse the hierarchical relationships between classes, which are not explicitly expressed in the two ontologies.

- Future work on this approach may include exploring constraints such as complex axioms in OWL, investigating which constraints are the most useful.

- It may be possible to improve the system with approaches such as learning-based techniques.

- It may be possible to improve the technique by using SKOS parser Language in order to deal with many ontology have been developed.

- It may be possible to improve the performance of the algorithm itself and of its implementation in order to deal with very large ontologies.

References

1. G. Antoniou and F.V. Harmelen, "*Web Ontology Language: OWL*", Presented at Handbook on Ontologies, 2004, pp.67-92.

2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P. Patel-Schneider, "*The Description Logic Handbook: Theory, Implementation and Applications*", Cambridge University Press, 2003.

3. F. Baader, I. Horrocks, and U. Sattler, "*Description Logics as Ontology Languages for the Semantic Web*", In Proceedings of Mechanising Mathematical Reasoning, 2005, pp.228-248.

4. D. Beckett, "*The Design and Implementation of the Redland RDF Application Framework*", In Proceedings of 10th International World Wide Web Conference, Hong Kong, May 200.

5. S. Bechhofer, C.A. Goble, and I. Horrocks, "*DAML+OIL is not Enough*", In Proceedings of SWWS, 2001, pp.151-159

6. T. Berners-Lee, J. Hendler, and O. Lassila, "*The Semantic Web*", Scientific Am, May 2001, pp. 34–43

7. M. Benerecetti, P. Bouquet, and C. Ghidini, "*Contextual Reasoning Distilled*", Presented at Journal Experimental and theoretical Artificial Intelligence (JETAI), 2000, pp.279-305.

8. S. Benslimane, A. Merazi, M. Malki and D. Bensaber, "*Ontology Mapping for Querying Heterogeneous Information Sources*", INFOCOMP (Journal of Computer Science), 2008.

9. S. Bechhofer, I. Horrocks, C.A. Goble, and R. Stevens, "*OilEd: a Reason-able Ontology Editor for the Semantic Web*", In Proceedings of Description Logics, 2001.

10. T. Bray, J. Paoli, and C.M. Sperberg-McQueen, "*Extensible Markup Language (XML)*", Presented at World Wide Web Journal, 1997, pp.27-66.

11. D. Brickley and R. Guha, *"Resource Description Framework (RDF) Schema specification*", 2000. http://www.w3.org/TR/RDF-schema.

12. P. B ouquet, L. Serafini and S. Zanobini, "*Semantic Coordination: A New Approach and An Application*", In Proceedings of ISWC2003, Florida, USA, 2003.

13. H. Bunke and J. Csirik, "*Parametric String Edit Distance and Its Application to Pattern Recognition*", IEEE Transactions on Systems, Man, and Cybernetics, vol. 25, pp. 202-206, 1995.

14. H. Chalupsky, "*OntoMorph: A Translation System for Symbolic Knowledge*", In Proceedings of KR, 2000, pp.471-482.

15. W.W. Cohen, P. Ravikumar, and S.E. Fienberg, "*A Comparison of String Distance Metrics for Name-Matching Tasks*", In Proceedings of II Web, 2003, pp.73-78.

16. O. Corcho and A. Gómez-Pérez, "*Solving Integration Problems of E-Commerce Standards and Initiatives through Ontological Mappings*", In Proceedings of IJCAI 2001 Workshop on E-Business & the Intelligent Web, Seattle, USA, 2001.

17. C. Curino, G. Orsi, and L. Tanca, "*X-SOM: A Flexible Ontology Mapper*", In Proceedings of 1st Intl. Workshop on Semantic Web Architectures For Enterprises, 2007.

18. M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein (editors), "*OWL Web Ontology Language 1.0 Reference*", http://www.w3.org/TR/owl-ref/, 2002.

19. J. De Bruijn, "*Using Ontologies. Enabling Knowledge Sharing and Reuse on the Semantic Web*", DERI Technical Report DERI-2003-10-29, October 2003, pp. 1-49.

20. N. Desai, A.U. Mallya, A.K. Chopra, and M.P. Singh, "*OWL-P: A Methodology for Business Process Development*", In Proceedings of AOIS, 2005, pp.79-94.

21. A. Doan, J. Madhavan, P. Domingos, and A.Y. Halevy, "*Learning to Map Between Ontologies on the Semantic Web*", In Proceedings of WWW, 2002, pp.662-673.

22. D. Dou, D. McDermott, and P. Qi, "*Ontology Translation on the Semantic Web*", Presented at on Data Semantics Journal, 3360:35–57, 2005.

23. H.H. Do and E. Rahm, "*COMA - A System for Flexible Combination of Schema Matching Approaches*", In Proceedings of VLDB, 2002, pp.610-621.

24. M. Ehrig, S. Staab, "*Efficiency of Ontology Mapping Approaches*", International Workshop on Semantic Intelligent Middleware for the Web and the Grid at ECAI 04, Valencia, Spain, August 2004.

25. M. Ehrig, "*Ontology Alignment: Bridging the Semantic Gap (Semantic Web and Beyond)*", New York, Springer, 2006.

26. M. Ehrig and S. Staab, "*QOM - Quick Ontology Mapping*", In Proceedings of International Semantic Web Conference, 2004, pp.683-697.

27. M. Ehrig and J. Euzenat, "*State of the Art on Ontology Alignment*", Knowledge Web Deliverable D2.2.3, University of Karlsruhe, 2004.

28. J. Euzenat and P. Shvaiko, "*Ontology Matching*", Springer-Verlag, Heidelberg (DE), 2007.

29. J. Euzenat and P. Valtchev, "*Similarity-Based Ontology Alignment in OWL-Lite*", In Proceedings of ECAI, 2004, pp.333-337.

30. J. Euzenat, D. Loup, M. Touzani, and P. Valtchev, "*Ontology Alignment with OLA*", In 3rd EON Workshop, 3rd Int. Semantic Web Conference, 2004, pp. 333–337.

31. D. Fensel, F.V. Harmelen, I. Horrocks, D.L. McGuinness, and P.F. Patel-Schneider, "*OIL: An Ontology Infrastructure for the Semantic Web*", Presented at IEEE Intelligent Systems, 2001, pp.38-45.

32. D. Fensel, "*Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*", Springer, 2001.

33. V. Haarslev and R. Moller, "*RACER System Description*", In Proceedings of IJCAR, 2001, pp.701-706.

34. F. van Harmelen, P.F. Patel-Schneider, and I. Horrocks (Editors), "*Reference Description of the DAML+OIL Ontology Markup Language*", http://www.daml.org/2000/12/reference.html, 2000.

35. I. Horrocks and P.F. Patel-Schneider, "*Reducing OWL Entailment to Description Logic Satisfiability*", Presented at Web Semantic Journal, 2004, pp.345-357.

36. W. Hu, Y. Zhao, D. Li, G. Cheng, H. Wu, and Y. Qu, "*Falcon-AO: Results for OAEI 2007*", In Proceedings of OM, 2007.

37. A. Isaac, L. van der Meij, S. Schlobach, S. Wang, "*An Empirical Study of Instance-based Ontology Matching*", In Proceedings of the 6th International Semantic Web Conference, Busan, Korea, 2007.

38. F. Giunchiglia and M. Yatskevich, "*Element Level Semantic Matching*", In Proceedings of the Meaning Coordination and Negotiation workshop at ISWC, (2004).

39. F. Giunchiglia, P. Shvaiko, and M. Yatskevich, "*S-Match: An Algorithm and An Implementation of Semantic Matching*", In Proceedings of Semantic Interoperability and Integration, 2005.

40. F. Giunchiglia, P. Shvaiko, and M. Yatskevich, "*Semantic Schema Matching*", In Proceedings of OTM Conferences (1), 2005, pp.347-365.

41. C. Ghidini and F. Giunchiglia, "*A Semantics for Abstraction*", In Proceedings of ECAI, 2004, pp.343-347.

42. A. Gomez-Perez and D. Manzano-Macho, "*A Survey of Ontology Learning Methods and Techniques*". deliverable 1.5, ontoweb project, 2003.

43. O. Gotoh, "*An Improved Algorithm for Matching Biological Sequences*", Presented at Journal of Molecular Biology, 162:705-708, 1982.

44. Y.R. Jean-Mary and M.R. Kabuka, "*ASMOV Results for OAEI 2007*", In Proceedings of OM, 2007.

45. J.J. Jiang and D.W. Conrath, "*Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy*", Presented at Journal CoRR, 1997.

46. Y. Kalfoglou and W.M. Schorlemmer, "*Ontology Mapping: The State of the Art*", In Proceedings of Semantic Interoperability and Integration, 2005.

47. Y. Kalfoglou and W.M. Schorlemmer, "*IF-Map: An Ontology-Mapping Method Based on Information-Flow Theory*", Presented at Journal Data Semantics, 2003, pp.98-127.

48. M.C.A. Klein and D. Fensel, "*Ontology Versioning on the Semantic Web*", In Proceedings of SWWS, 2001, pp.75-91.

49. F.N. Kepler, C. Paz-Trillo, J. Riani, M.M. Ribeiro, K.V. Delgado, L.N.D. Barros, and R. Wassermann, "*Classifying Ontologies*", In Proceedings of WONTO, 2006.

50. K. Kotis and G.A. Vouros, "*The HCONE Approach to Ontology Merging*", In Proceedings of ESWS, 2004, pp.137-151.

51. O. Lassila and R. Swick, "*Resource Description Framework (RDF) model and syntax specification*", 1999. http://www.w3.org/TR/REC-rdf-syntax.

52. P. Lambrix and H. Tan, "*A Tool for Evaluating Ontology Alignment Strategies*", Presented at Journal Data Semantics, 2007, pp.182-202.

53. C. Leacock and M. Chodorow, "*Combining Local Context and Wordnet Similarity for Word Sense Identification*", In WordNet: An Electronic Lexical Database, Christiane Fellbaum, MIT Press, 1998, pp. 265–283.

54. M. Li, M. Baker, "*The Grid: Core Technologies*", John Willey & Sons England (2005).

55. Y. Li, Q. Zhong, J. Li, and J. Tang, "*Result of Ontology Alignment with RiMOM at OAEI07*", In Proceedings of OM, 2007.

56. A. Maedche and S. Staab, "*Ontology Learning for the Semantic Web*", Presented at IEEE Intelligent Systems, 2001, pp.72-79.

57. D. Maynard and S. Ananiadou, "*A Linguistic Approach to Terminological Context Clustering*", Natural Language Pacific. Rim Symposium, 1999.

58. A. Marzal and E. Vidal, "*Computation of Normalized Edit Distance and Applications*", Presented at IEEE Transactionon Pattern Analysis and Machine Intelligence, 1993, pp.926-932.

59. M. Mao and Y. Peng, "*The PRIOR+: Results for OAEI Campaign 2007*", In Proceedings of OM, 2007.

60. B. McBride, S. Staab, R. Studer (eds.), "*The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS, in: The Handbook on Ontologies in Information Systems*", Springer-Verlag, 2003.

61. D.L. McGuinness, F. van Harmelen, "*OWL Web Ontology Language Overview*", Technical report, W3C, http://www.w3.org/TR/owl-features/, February 2004.

62. D.L. McGuinness, R. Fikes, J. Rice, and S. Wilder, "*An Environment for Merging and Testing Large Ontologies*", In Proceedings of KR2000, 2000, pp. 483-493.

63. D.L. McGuinness, R. Fikes, L.A. Stein, and J.A. Hendler, "*DAML-ONT: An Ontology Language for the Semantic Web*", In Proceedings of Spinning the Semantic Web, 2003, pp.65-93.

64. D.L. McGuinness, R. Fikes, J. Rice, and S. Wilder, "*The Chimaera Ontology Environment*", In Proceedings of the 17th National Conference on Artificial Intelligence (AAAI), 2000.

65. S. Melnik, H. Garcia-Molina, and E. Rahm, "*Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching*", In Proceedings of ICDE, 2002, pp.117-128.

66. G.A. Miller, "*WordNet: A Lexical Database for English*", presented at Commun. ACM, 1995, pp.39-41.

67. G.A. Miller and F. Hristea, "*WordNet Nouns:Classes and Instances*", Presented at Computational Linguistics, 2006, pp.1-3.

68. H. Mihoubi, A. Simonet, and M. Simonet, "*An Ontology Driven Approach to Ontology Translation*", In Proceedings of DEXA, 2000, pp.573-582

69. P. Mitra, G. Wiederhold, and M.L. Kersten, "*A Graph-Oriented Model for Articulation of Ontology Interdependencies*", In Proceedings of EDBT, 2000, pp.86-100.

70. A.E. Monge and C. Elkan, "*The Field Matching Problem: Algorithms and Applications*", In Proceedings of KDD, 1996, pp.267-270.

71. R. Neches, R. Fikes, T.W. Finin, T.R. Gruber, R.S. Patil, T.E. Senator, and W.R. Swartout, "*Enabling Technology for Knowledge Sharing*", Presented at AI Magazine, 1991, pp.36-56.

72. M. Nagy, M. Vargas-Vera, and E. Motta, "*DSSim - Managing Uncertainty on the Semantic Web*", In Proceedings of OM, 2007.

73. N.F. Noy, "*Semantic Integration: A Survey Of Ontology-Based Approaches*", Presented at SIGMOD Record, 2004, pp.65-70.

74. N. Noy and D. L. McGuinness, "*Ontology development 101: A guide to creating your first ontology*", Technical Report KSL-01-05, Stanford Medical Informatics, Stanford, 2001.

75. N.F. Noy and M.A. Musen, "*PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment*", In Proceedings of AAAI/IAAI, 2000, pp.450-455.

76. M. Paolucci, T. Kawamura, T.R. Payne, and K.P. Sycara, "*Semantic Matching of Web Services Capabilities*", In Proceedings of International Semantic Web Conference, 2002, pp.333-347.

77. P. F. Patel-Schneider, P. Hayes, and I. Horrocks, "*OWL Web Ontology Language: Semantics and Abstract Syntax*", W3C Recommendation, February 10 2004. (Available via http://www.w3.org/TR/2003/WD-owl-semantics-20030331/ , last visited March 2009).

78. E. Rahm, P.A. Bernstein, "*A Survey of Approaches to Automatic Schema Matching*", Presented at VLDB Journal, 2001, pp.334–350.

79. P. Resnik, "*Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language*", Presented at Journal Artificial Intelligence Resarch (JAIR), 1999, pp.95-130.

80. G. Salton, "*Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*"**,** Addison Wesley, New York, 1989.

81. M. Schorlemmer and Y. Kalfoglou, "*Progressive Ontology Alignment for Meaning Coordination: An Information-theoretic Foundation*", In Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, Utrecht, The Netherlands, 2005, pp. 737–744.

82. P. Shvaiko and J. Euzenat, *"A Survey of Schema-Based Matching Approaches*", Presented at Journal Data Semantics IV, 2005, pp.146-171.

83. M. Smith, C.Welty, D. McGuinness, "*OWL Web Ontology Language Guide*", http://www.w3.org/TR/2003/WD-owl-guide-20030331/.

84. J.F. Sowa, "*Knowledge Representation: Logical, Philosophical, and Computational Foundations*", Brooks Cole Publishing, Pacific Grove, 2000.

85. V. Spiliopoulos, A.G. Valarakos, G.A. Vouros, and V. Karkaletsis, "*SEMA: Results for the Ontology Alignment Contest OAEI 2007*", In Proceedings of OM, 2007.

86. G. Stumme and A. Maedche, "*FCA-MERGE: Bottom-Up Merging of Ontologies*", In Proceedings of IJCAI, 2001, pp.225-234.

87. X. Su and J.A. Gulla, "*An Information Retrieval Approach to Ontology Mapping*", Presented at Data & Knowledge Engineering, 2006, pp.47-69.

88. P. Tan, M. Steinbach and V. Kumar, "*Introduction to Data Mining*", Addison-Wesley Longman Publishing Co., Inc., Boston (2005).

89. D. Tidwell, "*Web Services-The Web's Next Revolution*", IBM Web Service Tutorial, 29 Nov. 2000, http://www-106.ibm.com/developerworks/edu/ws-dw-wsbasics-i.html.

90. D. Tsarkov and I. Horrocks, "*FaCT++ Description Logic Reasoner: System Description*", In Proceedings of the International Joint Conference on Automated Reasoning (IJCAR), volume 4130, pages 292–297, 2006.

91. M. Uschold and M. Gruninger, "*Ontologies: Principles, Methods and Applications*", Knowledge Engineering Review., vol. 11, no. 2, June 1996.

92. M. Uschold and M. Gruninger, "*Ontologies and Semantics for Seamless Connectivity*", Presented at SIGMOD Record, 2004, pp.58-64.

93. M. Uschold,. "*Where Are the Semantics in the Semantic Web?*", presented at AI Magazine, 2003, pp.25-36.

94. P. Wang and B. Xu, "*LILY: the Results for the Ontology Alignment Contest OAEI 2007*", In Proceedings of OM, 2007.

95. M. Wooldridge, "*An Introduction to Multiagent Systems*", John Wiley & Sons, LTD,Chichester, England, February 2002.

96. Ontology Alignment Evaluation Initiative (OAEI). [Online site]. http://oaei.ontologymatching.org/. [Accessed: January, 2009].

97. SecondString Project Page, http://secondstring.sourceforge.net/, 2006.

Appendix A

Example A:

<?xml version="1.0" encoding="UTF-8" ?>

- <rdf:RDF xmlns="http://knowledgeweb.semanticweb.org/heterogeneity/alignment" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

- <!--

Source: [http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/ontologies/sportSoccer.owl]
Target: [http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/ontologies/sportEvent.owl]
Resutls are produced with [Ontology Alignment]

-->

- <Alignment>

<xml>yes</xml>

<type>11</type>

<onto1>http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/ontologies/sportSoccer.owl</onto1>

<onto2>http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/ontologies/sportEvent.owl</onto2>

- <map>

- <Cell>

<entity1 rdf:resource="**http://sport.semanticweb.org/sport#name**" />

<entity2 rdf:resource="**http://protege.stanford.edu/kb#name**" />

<measure rdf:datatype="**xsd:float**">1.0</measure>

<relation>=</relation>

</Cell>

- <Cell>

<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Wing**" />

<entity2 rdf:resource="**http://protege.stanford.edu/kb#Wing**" />

<measure rdf:datatype="**xsd:float**">1.0</measure>

<relation>=</relation>

</Cell>

- <Cell>

<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Whistle**" />

<entity2 rdf:resource="**http://protege.stanford.edu/kb#Whistle**" />

<measure rdf:datatype="**xsd:float**">1.0</measure>

<relation>=</relation>

</Cell>

- <Cell>

<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Volley**" />

<entity2 rdf:resource="**http://protege.stanford.edu/kb#Volley**" />

<measure rdf:;datatype="**xsd:float**">1.0</measure>

<relation>=</relation>

</Cell>

- <Cell>

<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Trap**" />

<entity2 rdf:resource="**http://protege.stanford.edu/kb#Trap**" />

<measure rdf:datatype="**xsd:float**">1.0</measure>

<relation>=</relation>

</Cell>

- <Cell>

<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Trainer**" />

<entity2 rdf:resource="**http://protege.stanford.edu/kb#Trainer**" />

<measure rdf:datatype="**xsd:float**">1.0</measure>

<relation>=</relation>

</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Tournament**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Tournament**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Timekeeper**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Timekeeper**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Team**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Team**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Tackle**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Tackle**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Sweeper**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Sweeper**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Supporter**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Supporter**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Substitute**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Substitute**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Stopper**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Stopper**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>

<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Steal**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Steal**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Speaker**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Speaker**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Shoot**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Shoot**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Shielding**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Shielding**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Save**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Save**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Sanction**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Sanction**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Referee**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Referee**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#President**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#President**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Point**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Point**" />

<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Player**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Player**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Play**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Play**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Period**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Period**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Penetrate**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Penetrate**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Pass**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Pass**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Overtime**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#OverTime**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Overlap**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Overlap**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Nutmeg**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Nutmeg**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>

</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Net**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Net**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Midfielder**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Midfielder**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#MidfieldAnchor**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#MidfieldAnchor**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Midfield**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Midfield**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Match**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Match**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Marking**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Marking**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Lob**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Lob**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Line**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Line**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>

<entity1 rdf:resource="**http://sport.semanticweb.org/sport#League**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#League**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Kick**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Kick**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Juggling**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Juggling**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Inswinger**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Inswinger**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Hook**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Hook**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Hit**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Hit**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Header**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Header**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Handball**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#HandBall**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Halftime**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#HalfTime**" />

<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Hacking**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Hacking**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Goalkeeper**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Goalkeeper**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Forward**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Forward**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Feint**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Feint**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#End**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#End**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Encourage**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Encourage**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Dribble**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Dribble**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#DefensiveMidfielder**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#DefensiveMidfielder**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>

</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Cross**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Cross**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Club**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Club**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Clear**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Clear**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Charge**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Charge**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#CentralDefender**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#CentralDefender**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Center**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Center**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Cap**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Cap**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Breakaway**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Breakaway**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>

<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Boo**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Boo**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Block**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Block**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Begin**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Begin**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Beat**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Beat**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Ball**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Ball**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#AttackingMidfielder**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#AttackingMidfielder**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Attacker**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Attacker**" />
<measure rdf:datatype="**xsd:float**">1.0</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Score**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Scorer**" />
<measure rdf:datatype="**xsd:float**">0.9090909090909091</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Do_obstruction**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Obstruction**" />

<measure rdf:datatype="**xsd:float**">0.88</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Offside**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#BeOffside**" />
<measure rdf:datatype="**xsd:float**">0.875</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Do_handball**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#HandBall**" />
<measure rdf:datatype="**xsd:float**">0.8421052631578947</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Field**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Upfield**" />
<measure rdf:datatype="**xsd:float**">0.8333333333333334</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Penalty**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#PenaltyArc**" />
<measure rdf:datatype="**xsd:float**">0.8235294117647058</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Corner**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#CornerArc**" />
<measure rdf:datatype="**xsd:float**">0.8</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Run_upfield**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Upfield**" />
<measure rdf:datatype="**xsd:float**">0.7777777777777778</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Give_Sanction**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Sanction**" />
<measure rdf:datatype="**xsd:float**">0.7619047619047619</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Wall**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Ball**" />
<measure rdf:datatype="**xsd:float**">0.75</measure>
<relation>=</relation>

</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Object**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#GoalObject**" />
<measure rdf:datatype="**xsd:float**">0.75</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Call**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Ball**" />
<measure rdf:datatype="**xsd:float**">0.75</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Back**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Sack**" />
<measure rdf:datatype="**xsd:float**">0.75</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Action**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#BodyAction**" />
<measure rdf:datatype="**xsd:float**">0.75</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Goalkeeper_action**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Goalkeeper**" />
<measure rdf:datatype="**xsd:float**">0.7407407407407407</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Be_Offside**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#BeOffside**" />
<measure rdf:datatype="**xsd:float**">0.7368421052631579</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#forname**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#name**" />
<measure rdf:datatype="**xsd:float**">0.7272727272727273</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Time**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#EndTime**" />
<measure rdf:datatype="**xsd:float**">0.7272727272727273</measure>
<relation>=</relation>
</Cell>
- <Cell>

<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Stop**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Stopper**" />
<measure rdf:datatype="**xsd:float**">0.7272727272727273</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Official**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#OfficialAction**" />
<measure rdf:datatype="**xsd:float**">0.7272727272727273</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Linesman**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#LinesmanAction**" />
<measure rdf:datatype="**xsd:float**">0.7272727272727273</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Kickoff**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Kick**" />
<measure rdf:datatype="**xsd:float**">0.7272727272727273</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Do_foul**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Foul**" />
<measure rdf:datatype="**xsd:float**">0.7272727272727273</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Penetrating_pass**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#PenetratingPass**" />
<measure rdf:datatype="**xsd:float**">0.7096774193548387</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Side_tackle**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Tackle**" />
<measure rdf:datatype="**xsd:float**">0.7058823529411765</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#OtherPlayer**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Player**" />
<measure rdf:datatype="**xsd:float**">0.7058823529411765</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Center_line**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Center**" />

<measure rdf:datatype="**xsd:float**">0.7058823529411765</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Center_kick**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Center**" />
<measure rdf:datatype="**xsd:float**">0.7058823529411765</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Center_Spot**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Center**" />
<measure rdf:datatype="**xsd:float**">0.7058823529411765</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Back_tackle**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Tackle**" />
<measure rdf:datatype="**xsd:float**">0.7058823529411765</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Back_header**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Header**" />
<measure rdf:datatype="**xsd:float**">0.7058823529411765</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#match_referee**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#referee**" />
<measure rdf:datatype="**xsd:float**">0.7</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#team_substitute**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#substituteWhom**" />
<measure rdf:datatype="**xsd:float**">0.6896551724137931</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Throw_in**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#ThrowIn**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Stoppage**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Stopper**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>

</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Sending_off**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#SendingOff**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Referee_action**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Referee**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Red_Card**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Card**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Place**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Play**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Penalty_Arc**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#PenaltyArc**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Offside_trap**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#BeOffside**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Goal**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#GoalLine**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Front_tackle**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Tackle**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>

<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Front_header**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Header**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Flick_header**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Header**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Do_Professional_foul**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#ProfessionalFoul**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Dangerous_play**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#DangerousPlay**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Chip_shoot**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Shoot**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Area**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Break**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Advantage_rule**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#AdvantageRule**" />
<measure rdf:datatype="**xsd:float**">0.6666666666666666</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Hold_opponent**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#HoldOpponent**" />
<measure rdf:datatype="**xsd:float**">0.64</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Penalty_spot**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#PenaltyArc**" />

<measure rdf:datatype="**xsd:float**">0.6363636363636364</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Penalty_kick**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#PenaltyArc**" />
<measure rdf:datatype="**xsd:float**">0.6363636363636364</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Penalty_Area**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#PenaltyArc**" />
<measure rdf:datatype="**xsd:float**">0.6363636363636364</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Caution**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#FootballCaution**" />
<measure rdf:datatype="**xsd:float**">0.6363636363636364</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Player_Action**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Player**" />
<measure rdf:datatype="**xsd:float**">0.631578947368421</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Diving_header**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Diving**" />
<measure rdf:datatype="**xsd:float**">0.631578947368421</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Corner_Arc**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#CornerArc**" />
<measure rdf:datatype="**xsd:float**">0.631578947368421</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Center_Circle**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Center**" />
<measure rdf:datatype="**xsd:float**">0.631578947368421</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#club_name**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#name**" />
<measure rdf:datatype="**xsd:float**">0.6153846153846154</measure>
<relation>=</relation>

</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Wall_pass**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Pass**" />
<measure rdf:datatype="**xsd:float**">0.6153846153846154</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Push_pass**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Pass**" />
<measure rdf:datatype="**xsd:float**">0.6153846153846154</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Long_pass**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Pass**" />
<measure rdf:datatype="**xsd:float**">0.6153846153846154</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Lead_pass**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Pass**" />
<measure rdf:datatype="**xsd:float**">0.6153846153846154</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Goal_kick**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Kick**" />
<measure rdf:datatype="**xsd:float**">0.6153846153846154</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Goal_Line**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Line**" />
<measure rdf:datatype="**xsd:float**">0.6153846153846154</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Free_kick**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Kick**" />
<measure rdf:datatype="**xsd:float**">0.6153846153846154</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Foot_trap**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Trap**" />
<measure rdf:datatype="**xsd:float**">0.6153846153846154</measure>
<relation>=</relation>
</Cell>
- <Cell>

<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Drop_kick**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Kick**" />
<measure rdf:datatype="**xsd:float**">0.6153846153846154</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Drop_ball**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Ball**" />
<measure rdf:datatype="**xsd:float**">0.6153846153846154</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Chip_pass**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Pass**" />
<measure rdf:datatype="**xsd:float**">0.6153846153846154</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Sliding_tackle**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Tackle**" />
<measure rdf:datatype="**xsd:float**">0.6</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Person**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#FootballPerson**" />
<measure rdf:datatype="**xsd:float**">0.6</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Dive**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#Diving**" />
<measure rdf:datatype="**xsd:float**">0.6</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Corner_kick**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#CornerArc**" />
<measure rdf:datatype="**xsd:float**">0.6</measure>
<relation>=</relation>
</Cell>
- <Cell>
<entity1 rdf:resource="**http://sport.semanticweb.org/sport#Corner_Area**" />
<entity2 rdf:resource="**http://protege.stanford.edu/kb#CornerArc**" />
<measure rdf:datatype="**xsd:float**">0.6</measure>
<relation>=</relation>
</Cell>
</map>
</Alignment>
</rdf:RDF>

Example B:

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
 <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasMother">
  <owl:sameAs>
   <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasMother"/>
  </owl:sameAs>
 </owl:Class>
 <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasMother">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasMother"/>
 </owl:Class>
 <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasFather">
  <owl:sameAs>
   <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasFather"/>
  </owl:sameAs>
 </owl:Class>
 <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#shoesize">
  <owl:sameAs>
   <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#shoesize"/>
  </owl:sameAs>
 </owl:Class>
 <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Animal">
  <owl:sameAs>
   <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Animal"/>
  </owl:sameAs>
 </owl:Class>
 <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Woman">
  <owl:sameAs>
   <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Woman"/>
  </owl:sameAs>
 </owl:Class>
 <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Person">
  <owl:sameAs>
   <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Person"/>
  </owl:sameAs>
 </owl:Class>
 <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Man">
  <owl:sameAs>
   <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Man"/>
  </owl:sameAs>
 </owl:Class>
 <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasSpouse">
  <owl:sameAs>
   <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasSpouse"/>
  </owl:sameAs>
```

```
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Man">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Man"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasChild">
  <owl:sameAs>
    <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasChild"/>
  </owl:sameAs>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#biologicalMotherOf">
  <owl:sameAs>
    <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#biologicalMotherOf"/>
  </owl:sameAs>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Animal">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Animal"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Woman">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Woman"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasSpouse">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasSpouse"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#biologicalMotherOf">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#biologicalMotherOf"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#shirtsize">
  <owl:sameAs>
    <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#shirtsize"/>
  </owl:sameAs>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#spouseOf">
  <owl:sameAs>
    <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#spouseOf"/>
  </owl:sameAs>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Female">
  <owl:sameAs>
    <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Female"/>
  </owl:sameAs>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasAncestor">
  <owl:sameAs>
    <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasAncestor"/>
  </owl:sameAs>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasFather">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasFather"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#age">
```

```
  <owl:sameAs>
    <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#age"/>
  </owl:sameAs>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Person">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Person"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasFriend">
  <owl:sameAs>
    <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasFriend"/>
  </owl:sameAs>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasParent">
  <owl:sameAs>
    <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasParent"/>
  </owl:sameAs>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasAncestor">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasAncestor"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#shirtsize">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#shirtsize"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#age">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#age"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#spouseOf">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#spouseOf"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Male">
  <owl:sameAs>
    <owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Male"/>
  </owl:sameAs>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#shoesize">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#shoesize"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Male">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Male"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasParent">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasParent"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasFriend">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasFriend"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#Female">
  <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#Female"/>
</owl:Class>
<owl:Class rdf:about="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsB.owl#hasChild">
```

```
    <owl:sameAs rdf:resource="http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl#hasChild"/>
  </owl:Class>
</rdf:RDF>
```

Appendix B

```
package Ontology.Alignment.System.OntAliSystem;

import Ontology.Alignment. System.semanticFeature.*;

import Ontology.Alignment.System. matcher.*;

import Ontolog.Alignment.System.ontologyMatcher.*;

import Ontology.Alignment.System.until.*;

import Ontolog.Alignment.System.aggregator.*;

import java.util.*;

import java.net.*;

import java.io.*;

import Ontolog. Alignment.System.Exception.UnrecognisedTypeException;

public class OntAliSystem {

	/*
	* Main entry point for our System, it can be invoked by instantiating * a new
	instance or calling the static methods
	 */
public static String runONTALI (String[] pars) throws Exception {

				OntologyAlignment onta = new OntologyAl
ignment(pars[0], pars[1]);

				String[] machs = pars[2].split(" |,|\t");

				for (String m : machs) Accumulate.print(System.out,
"[OntAliSystem]", "Matcher " + m, true);

				onta.setMatchers(machs);

				double[] weights = new double[machs.length];

				for (int i = 0; i < machs.length; i++) {
		weights[i] = 1;

				}

				if (pars[3] != null) {

					String[] st = pars[3].split(" |,|\t");

					for (String s : st)
Accumulate.print(System.out, "\t\t", "Weight " + s, true);

					for (int i = 0; i < st.length; i++) {

						Double w = new
Double(st[i].trim());

						if (w.isNaN()) {

							weights[i]
= 1;

						} else {

							weights[i]
= w.doubleValue();

						}

					}

				}

		/** Static method that enables running OntologyAlginmentfrom
command line.
		*
*/

		onta.setMatcherWeights(machs, weights); /* seting the weight for
each mtchers */

				onta.setOutputOption(false);

				onta.setSubject(0);

				double thre;

				try		{

					thre =
Double.parseDouble(pars[4].trim());

					Accumulate.print(System.out,
"[OntAliSystem]", "threshold " + thre, true);

/* set the  threshold */

				} catch (NumberFormatException nfe) {

					thre = 0;

					Accumulate.print(System.out,
"[OntAliSystem]", "reset threshold ", true);

				}

				Vec<String> outputformatcherss = new
Vec<String>(); /* set the output format*/

				String[] ofnames = pars[5].split(" |,|\t");

				outputformatcherss.add("0");

				for (String ofname : ofnames) {

					if
(ofname.trim().equalsIgnoreCase("XMLFORMAT")) {


		outputformatcherss.add("3");
		}

					if
(ofname.trim().equalsIgnoreCase("OWL")) {


		outputformatcherss.add("1");
		}

							}

				String[] outputformatarray = new
String[outputformatcherss.size()];

				for (int i = 0; i < outputformatcherss.size(); i++) {

					outputformatarray[i] =
outputformatcherss.elementAt(i);

				}

				return onta.sorted(thre, outputformatarray);

		}

public static void runONTALIContamandLine (String[] pars) throws Exception {

/**Static method that allows running OntoALI with predefined parametereters.
Parametereters accepted to *OntAliSystem.run(String[]) are in a exacting order
as: < source ontology >, < target ontology >, < matchers * * > < weights >, <
threshold > Outputs are written to the paradigm output device, defaulting
matching *include both concepts and properties.
*/

OntologyAlignment onta = null;

String sourceOnt = null, targetOnt = null, matcherss = null, out = null, threshold =
null, weight = null, aggs = null; boolean sourceOntok = false, targetOntok = false,
outok = false, mok = false, thok = false, xmlFormatok = false, htmlFormatok =
false, owlok = false, weightok = false, aggok = false, conceptok = false, propok =
false, lowcostok = false, sook = false;

				for (int i = 0; i < pars.length; i++) {

Accumulate.print(System.out, "\t", "parametereter " + (i+1) + ": " + pars[i], true);
/* set the out put format*/

					if (pars[i].equals("-s")) {
```

```
                                                sourceOnt
= pars[i+1];

                sourceOntok = true;

                                }

                                if (pars[i].equals("-t")) {

                                                targetOnt
= pars[i+1];

                targetOntok = true;

                                }

                                if (pars[i].equals("-m")) {

                                                matcherss
= pars[i+1];

                                                mok =
true;

                                }

                                if (pars[i].equals("-xmlFormat")) {

                                                if (
(!pars[i+1].startsWith("-")) && (!pars[i+1].startsWith("+")) ) {

                                                out =
pars[i+1];

                                                }

                                                outok = true;

                                                xmlFormatok = true;

                                }

                                if (pars[i].equals("-owl")) {

                                                if (
(!pars[i+1].startsWith("-")) && (!pars[i+1].startsWith("+")) ) {

                                                out =
pars[i+1];

                                                }

                                                outok = true;

                                                owlok = true;

                                }

                        if (pars[i].equals("-htmlFormat")) {

                                                if (
(!pars[i+1].startsWith("-")) && (!pars[i+1].startsWith("+")) ) {

                                                out =
pars[i+1];

                                                }

                                                outok = true;

                                                htmlFormatok = true;

                                }

                                if (pars[i].equals("-w")) {

                                                weight = pars[i+1];

                                                weightok = true;

                                }

                                if (pars[i].equals("-a")) {

                                                aggs = pars[i+1];

                                                aggok = true;

                                }

                                if (pars[i].equals("-T")) {

                                                threshold = pars[i+1];

                                                thok = true;

                                }

                                if (pars[i].equals("+c")) {

                                                conceptok = true;

                                }

                                if (pars[i].equals("+p")) {

                                                propok = true;

                                }

                                if (pars[i].equals("+cp")) {

                                                propok = true;

                                                conceptok = true;

                                }

                                if (pars[i].equals("-c")) {

                                                conceptok = false;

                                }

                                if (pars[i].equals("-p")) {

                                                propok = false;

                                }

                                if (pars[i].equals("-l")) {

                                                lowcostok = true;

/*Sets whether running the matching algorithms with low resource cost */

                                }

                                if (pars[i].equals("-so")) {

                                                sook = true;

                                }

                        }

                        if (!sourceOntok || !targetOntok) {

                                                throw new
Exception("[Ontontaapper.RUN] either Source or Target ontology is not given");

                        } /* invoke  the matchers*/

                        onta = new OntologyAlignment(sourceOnt,
targetOnt);

                        String[] machs;

                        if (!mok) {

                                                Accumulate.print(System.out,
"[OntAliSystem]", "default matchers are used", true);
```

```
                machs = new String[] {"Jaro",
"MongeElkan",  "Heureistic Matcher "};

            } else {

                machs = matcherss.split(" |,|\t");

                for (String m : machs)
Accumulate.print(System.out, "[OntAliSystem]", "Matcher " + m, true);

            }

/**

 * A Synonym of a term in the registry.

 */

public class Synonym extends AbsttEntity {

  /**

   * Creates an Classe of this class.

    * @param the registry

   * @param synonym  Classe representing the term synonym

   */

  public Synonym(Registry registry, Classe synonym) {

    super(registry, synonym);

  }

  /**

   * Returns the URI of the type of this object.

   * @return the URI of the object type

   */

  protected String findTypeURI() {

  }

  /**

   * Returns the terms that this synonym references.

   * @return   the set of terms referenced by this synonym

   */

  public Set findReferTerms(){

    Set result=new HashSet();

    Prop references=m_entity.findOIModel().findProp( ());

    Iterator iterator=m_entity.findFromPropValues(refe).iterator();

    while (iterator.hasNext()) {

      Classe Classe=(Classe)iterator.next();

      result.add(m_registry.findTerm(Classe.findURI()));

    }

    return result;

  }
```

```
        /** Sets the weights of different matchers.
         * @parameter names String [], names of columns;
         * @parameter weights double [], values of weights.
         *
         **/
        onta.setMatchers(machs);

        Accumulate.print(System.out, "\t\t", "number of matchers: " +
machs.length, true);

        double[] weights = new double[machs.length];

        if (weightok) {

        String[] st = weight.split(" |,|\t");

        for (String s : st) Accumulate.print(System.out, "\t\t", "Weight " + s,
true);

            for (int i = 0; i < st.length; i++) {

                Double w = new Double(st[i].trim());

                    if (w.isNaN()) {

                        weights[i]
= 1;

                    } else {

                        weights[i]
= w.doubleValue();

                    }

                }

                if (st.length < machs.length) {

                    for (int i = st.length; i <
machs.length; i++) {

                        weights[i]
= 1;

                    }

                }

            } else {

                for (int i = 0; i < weights.length ; i++) {

                    weights[i] = 1;

                }

            }

            onta.setMatcherWeights(machs, weights);

            if (outok) {

                onta.setOutputFile(out);

            }

            onta.setOutputOption(sook);

            onta.setLowcostOption(lowcostok);

            double thre = 0;

            if (aggok) {

                onta.setAggregator(aggs); /* invoke
the aggregation algorithms  & Sets the aggregator name*/

            }

            if (conceptok && propok) {

                onta.setSubject(0);
```

```
                } else if (conceptok) {
                        onta.setSubject(1);
                } else if (propok) {
                        onta.setSubject(2);
                }
        /** Returns the final class matching results with
         */
                if (thok) {
                        try             {
        Accumulate.print(System.out, "[OntAliSystem]", "read threshold [" + threshold + "]", true);
                                thre = Double.parseDouble(threshold.trim());
        Accumulate.print(System.out, "[OntAliSystem]", "threshold " + thre, true);
                                } catch (NumberFormatException nfe) {
                                thre = 0;
        Accumulate.print(System.out, "[OntAliSystem]", "reset threshold ", true);
                                }
                }
                Vec<String> outputformatcherss = new Vec<String>();
                Vec<String> ofnames = new Vec<String>();
                outputformatcherss.add("0");
                if (xmlFormatok) {
                        outputformatcherss.add("3");
                }
                if (owlok) {
                        outputformatcherss.add("1");
                }
                String[] outputformatarray = new String[outputformatcherss.size()];
                for (int i = 0; i < outputformatcherss.size(); i++) {
                        outputformatarray[i] = outputformatcherss.elementAt(i);
                }
                onta.sorted(thre, outputformatarray);
        }
        private UsefulFeature loadOntology(String uri1, String uri2) {
                OWLUsefulMemFeature(new URI(uri1), new String[]{uri2});
                        }
                        if (maluri1) {
        Accumulate.print(System.out, "[OntAliSystem]", " input URI " + uri1 + " cannot be recognised", true);
        Accumulate.print(System.out, "     ", " feature extractor will try to load ontologies as local file", true);
        Accumulate.print(System.out, "    ", "\"\" will be assigned as the base URI which might have unexpected consequences", true);
                                if (maluri2) {
                                        ont = new OWLUsefulMemFeature(uri1, "", null);
                                } else {
                                        ont = new OWLUsefulMemFeature(uri1, "", new String[]{uri2});
                                }
                        }
                        if (ont == null) {
                                throw new Exception("[ONTA] ontology " + uri1 +" cannot be loaded");
                        }
                } catch (Exception ex) {
                        ex.printStackTrace();
                }
                return ont;
        }
package Ontology. Alignment. System.matcher;
public class SimpleEDNameMatcher extends NameMatcher {
 /** which is compute the minimum number of token insertions, deletions and substitutions *required to transform one string into another. Compute the similarity between names of  classes.
*/
public SimpleEDNameMatcher(String firstSource, String secondSource) {
/* This class computes the edit distance between two strings */
                super(firstSource, secondSource);
        }
    public void setOperandOne(String source) {
/** Sets first Source.
     *
     * @parameter source String.
     */
                name1 = source;
                lengthOne = name1.length();
        }
    public void setOperandTwo(String source) {
                name2 = source;
                lengthTwo = name2.length();
/** Sets second Source.
     *
     * @parameter source String.
     */
        }
/*Computes the similarity of two strings using Levenshtein string distance metric*/
```

```
		private int MiniMatUnm (int a, int b, int c) {

// find minimum of three values

			int mi = a;

			if (b < mi) {

				mi = b;

			}

			if (c < mi) {

				mi = c;

			}

			return mi;

		}

		protected double dontaatching() {

// calculate Levenshtein distance
  /**
 * Computes the similarity of two strings by using Levenshtein string distance
metric.
	* @return double, similarity
	*/

			int r = 0;

			double res;

			try		{

				if ((name1!=null) && (name2!=null)) {

					r =
LevenshteinDistance(name1, name2);

				} else {

					throw new
Exception("Matching Operands are not initialised");

				}

			} catch (Exception e) {

				e.printStackTrace();

			}

			if (lengthTwo > lengthOne) {

				res = 1 - (float)r/lengthTwo;

			} else {

				res = 1 - (float)r/lengthOne;

			}

			return res;

/** Computes the LevenshteinDistance of two strings.
    * @parameter s String,
    * @parameter t String,
     * @return int, String distance.
      */

		}

	protected int LevenshteinDistance (String s, String t) {

			int d[][];

			int n;

			int m;

			int i;

			int j;

			char s_i;

			char t_j;

			int cost;

// Step 1

n = s.length (); m = t.length ();

if (n == 0) {

		return m;

}

if (m == 0) {

		return n;

}

d = new int[n+1][m+1];

// Step 2

for (i = 0; i <= n; i++) {

		d[i][0] = i;

}

for (j = 0; j <= m; j++) {

		d[0][j] = j;

}

// Step 3

for (i = 1; i <= n; i++) {

		s_i = s.charAt (i - 1);

// Step 4

		for (j = 1; j <= m; j++) {

			t_j = t.charAt (j - 1);

// Step 5

			if (s_i == t_j) {

				cost = 0;

			} else {

				cost = 1;

			}

// Step 6

			d[i][j] = MiniMatUnm
(d[i-1][j]+1, d[i][j-1]+1, d[i-1][j-1] + cost);

		}

	}

// Step 7

	return d[n][m];

}
```

```
                public static void main (String[] args) {

                SimpleEDNameMatcher ed = new
SimpleEDNameMatcher(args[0], args[1]);

                System.out.println("result:
"+ed.getMatchingResult());

        }

}



package Ontology. Alignment. System. matcher;

import similaritymetrics.Soundex;

/* this matcher invokes Soundex distance methods provided by similarity
metrics, computing the phonetic similarity between names from their
corresponding Soundex codes  */

public class SoundexNameMatcher extends NameMatcher {

    /*
     * @parameter First source String
     * @parameter second source String
     */


 public SoundexNameMatcher (String firstSource, String secondSource) {

/*This class computes the SoundEx distance between two strings */

super(firstSource, secondSource);

        }

/** @parameter First source String, string one;
 * @parameter second source String, string two;
 * @parameter wordy boolean, true if verbose clarification is required; false
otherwise.
     */


public SoundexNameMatcher (String firstSource, String secondSource, boolean
wordy) {

        this(firstSource, secondSource);



        }

/** Sets string one.
   * @parameter First source String.
    */


    public void setOperandOne(String source) {

                name1 = source;

        }/** Sets string two.
    * @parameter second source String.
     */


    public void setOperandTwo(String source) {

                name2 = source;

        }



/*this mehod boolean, returns true if verbose explanation is required; false
otherwise */

  // Compute Soundex Similarity
  /**
  * Computes the Soundex distance between two strings.
  * @return double, similarity of two strings based on soundex distance.
     */

      protected double dontaatching() {

        double res = -1;

        Soundex sdex = new Soundex();

        try     {

                if ((name1 != null) && (name2 != null)) {

                        res = sdex.getSimilarity(name1,
name2);

                } else {

                throw new Exception("Contaparing operands are not
initialised");

                        }

                } catch (Exception e) {

                        e.printStackTrace();

                }

                return res;

        }

                public static void main (String[] args) {

                SoundexNameMatcher snm = new
SoundexNameMatcher(args[0], args[1]);

/* returen the similsrty value which Computed by  the Soundex between two
strings */

                System.out.println("[Soundex Distance]
"+snm.getMatchingResult());

        }

}



package Ontology. Alignment. System.matcher;

import net.didion.jwnl.JWNL;

import net.didion.jwnl.JWNLException;

import net.didion.jwnl.dictionary.Dictionary;

import net.didion.jwnl.data.*;

import java.io.*;

import java.util.Vec;

import java.util.Iterator;

importutil.*;

public class WNMatcher extends WNNameMatcher {

/*  it is a matcher using WordNet thesures  in order to find the relation between
the Name of classes by using thire noune so this measure determines the
meaning of the terms, which includes information such as synonyms. In this
phase, tring to find a common element in the synsets of two names */

protected String smatcher = Accumulate.JARO;

public WNMatcher(String firstSource, String secondSource) {

                super(firstSource, secondSource);

/*    This class computes the similarity of two strings by comparing their
WordNet synsets */

        }

        public WNMatcher(String firstSource, String secondSource, String
pos) {
```

```
                super(firstSource, secondSource, pos);

// String Distance matcher, Accumulate.JARO as default

}

        public void setStringMatcherType (String stringmatcher) {

                smatcher = stringmatcher;

        }

        protected double dontaatching () {

/*This class computes string similarity using both string distance based modules and WordNet based module */

                double globalsim = 0;

        if (Accumulate.GlobalFlags.Lowcost) {

        Vec synOfSource = getSynonymachsOfWord(name1);

        Vec synOfTarget = getSynonymachsOfWord(name2);

        Accumulate.print(System.out, "[WNP]", "contaparing " + synOfSource + "\n\n " + synOfTarget, false);

                double sim = 0;

                        try             {

        Class mc = ONTOLOGYALSLosder.load("Ontology. Alignment. System.matcher." + smatcher + "Matcher");

                                outter: for (Iterator si = synOfSource.iterator(); si.hasNext(); ) {

                                        String ssym = (String)si.next();

                                        for (Iterator ti = synOfTarget.iterator(); ti.hasNext(); ) {


        String tsym = (String)ti.next();

        NameMatcher nm = (NameMatcher)mc.getConstructor(String.class, String.class).newInstance(ssym, tsym);


        double local = nm.getMatchingResult();

        if (local == 1) {


                sim = local;

                break outter;

        } else if (local > sim) {


                sim = local;
                        }

                                        }

                                }

Accumulate.print(System.out, "[WNP]", name1 + " " + name2 + " result " + sim + "\n", false);

                        } catch (Exception ex) {

                                ex.printStackTrace();

                        }

                        return sim;

                } else {

                        for ( String sourceOntsub : sourceOntsubs ) {

                                double maxwordsim = 0;

                                for ( String targetOntsub : targetOntsubs ) {

                                        Vec synOfSource = getSynonymachsOfWord(sourceOntsub);

                                        Vec synOfTarget = getSynonymachsOfWord(targetOntsub);

Accumulate.print(System.out, "[WNP]", "contaparing " + synOfSource + "\n\n " + synOfTarget, false);

                                        double localsim = 0;

                                        try
        {

        Class mc = ONTOLOGYALSLosder.load("Ontology. Alignment. System.matcher." + smatcher + "Matcher");

                                outter: for (Iterator si = synOfSource.iterator(); si.hasNext(); ) {

                                        String ssym = (String)si.next();

                                for (Iterator ti = synOfTarget.iterator(); ti.hasNext(); ) {

                                        String tsym = (String)ti.next();

NameMatcher nm = (NameMatcher)mc.getConstructor(String.class, String.class).newInstance(ssym, tsym);

        Accumulate.print(System.out, "[WNP]", targetOntsub + " " + sourceOntsub + " result " + localsim + "\n", false);


        } catch (Exception ex) {


        ex.printStackTrace();

                                        }

                                        if (localsim > maxwordsim) {


        maxwordsim = localsim;

                                        }

                                }

                                globalsim += maxwordsim;

        Accumulate.print(System.out, "[WNP]", "globalsim " + globalsim + "\n", false);

                        }

        double finalsim = globalsim/( Math.max(targetOntsubs.length, sourceOntsubs.length) + sourceOntswcount + targetOntswcount );

        Accumulate.print(System.out, "[WNP]", "sim(" + name1 + ", " + name2 + ")= " + finalsim + "\n", false);

                        return finalsim;

                }

        }

        public static void main (String[] args) {
```

```
		WNMatcher wpn = new WNMatcher(args[0], args[1]);

		System.out.println("[WNP] "+wpn.getMatchingResult());

}

}

package Ontology.Alignment.System.ontologyMatcher;

import java.util.*;

import java.io.*;

import java.net.*;

import  Ontolog.Alignment.System.semanticFeature.*;

import Ontology.Alignment.System.until.*;

import Ontology.Alignment.System. matcher.*;

/*  it is a matcher using WordNet thesures  in order to find the relation between
the Name of classes by using thire noune so this measure determines the
meaning of the terms, which includes information such hyponyms. In this phase,
tring to find a common element in the synsets of two names */

public WNSynHypNameMatcher (String source1, String uri1, String source2,
String uri2) {

/**
 * This class invokes a string distance based name matcher and an instance of
WNSynHypNameMatcher as the underlying ontology name matchers.
 */
				super(source1, ns1, source2, ns2);

		}

/**  Equivalent to WNSynHypNameMatcher(source1, uri1, source2, uri2,
Accumulate.JARO)
	*
	* @parameter source1 String, filename of the source ontology
	* @parameter uri1 String,   base URL of the source ontology
	* @parameter source2 String, filename of the target ontology
	* @parameter uri2 String,   base URL of the target ontology
	*/
public WNSynHypNameMatcher (String uri1, String uri2) {

				super(uri1, uri2);

/*This class computes the distance between two strings based on WordNet
hierarchical structur */

/*This class computes string similarity using both string distance based modules
and WordNet based module */

		}

		public WNSynHypNameMatcher (URI uri1, URI uri2) {

				super(uri1, uri2);

		}

		public WNSynHypNameMatcher (String uri1, String uri2, String
type) {

				super(uri1, uri2, type);

	/** String matcher, JaroNameMatcher by default.
	 *
	 */
}

public WNSynHypNameMatcher (URI uri1, URI uri2, String type) {

				super(uri1, uri2, type);

		}

/** Constructor
	*  Equivalent to WNSynHypNameMatcher (uri1, uri2, Accumulate.JARO)
	*
	* @parameter uri1 String, uri of the source ontology
	* @parameter uri2 String, uri of the target ontology
	*/
public WNSynHypNameMatcher (UsefulFeature sx1, UsefulFeature sx2) {

		this(sx1, sx2, Accumulate.JARO);

}

/** Constructor
	*  Equivalent to WNSynHypNameMatcher (uri1, uri2, Accumulate.JARO)
	*
	* @parameter uri1 URI, uri of the source ontology
	* @parameter uri2 URI, uri of the target ontology
	*/
public WNSynHypNameMatcher (UsefulFeature sx1, UsefulFeature sx2, String
type) {

/** Constructor.
	*
	* @parameter uri1 URI, uri of the source ontology
	* @parameter uri2 URI, uri of the target ontology
	* @parameter type String, type of NameMatcher
	*/
				super(sx1, sx2);

				matchertype = type;

		loadMatcher();

}

/** Const
  * Equivalent to WNSynHypNameMatcher (sx1, sx2, Accumulate.JARO).
*
* @parameter sx1 UsefulFeature, source ontology wrapped with a UsefulFeature
 * @parameter sx2 UsefulFeature, target ontology wrapped with a UsefulFeature
		*
		*/
protected void loadWNNameMatcher() {

		try		{

Accumulate.print(System.out, "[OntologyWNPlus]", "loading WordNet Plus Name
Matcher", false);

wnmc = ONTOLOGYALSLosder.load("Ontology. Alignment. System.matcher." +
Accumulate.WNPN + "Matcher");

				} catch (Exception ex) {

					ex.printStackTrace();

				}

		}

   public static void main( String[] args ){

/** Returns class matching candidates.
	* @return Iterator, iterator over matching classes.
	*/


				try		{

					FileOutputStream fos = new
FileOutputStream("wnnameresult.txt");

					PrintStream pw =  new PrintStream
(fos);

WNSynHypNameMatcher wnm = new WNSynHypNameMatcher(new
URI(args[0]), new URI(args[1]));

				pw.println("[classes]"); // output matching classes

					for (Iterator i =
wnm.getMatchingClassesWithSco(0); i.hasNext(); ) {

						MatchingUnit o =
(MatchingUnit)i.next();

						String sourceOnt =
o.getOperandOne();

						String targetOnt =
o.getOperandTwo();

						double d = o.getSco();
```

```
                                                if (d >= 0.5) {

                pw.println(sourceOnt + " " + targetOnt + " " + d);

                Accumulate.print(System.out, "[OntologyWNPlus]", sourceOnt + " "
+ targetOnt + " " + d, false);

                                        }

                                        }

                                pw.println("[Prop]"); // output matching properties

                                        for (Iterator i =
wnm.getMatchingPropWithSco(0); i.hasNext(); ) {

                                                MatchingUnit o =
(MatchingUnit)i.next();

                                                String sourceOnt =
o.getOperandOne();

                                                String targetOnt =
o.getOperandTwo();

                                                double d = o.getSco();

                                                if (d >= 0.5) {

                                                pw.println(sourceOnt +
" " + targetOnt + " " + d);

                Accumulate.print(System.out, "[OntologyWNPlus]", sourceOnt + " "
+ targetOnt + " " + d, false);

                                                }

                                        }

                                        pw.close();

                                        fos.close();

                } catch (Exception ex) {

      /** Returns class matching candidates whose confidence value is greater
than threshold
       */

                                        ex.printStackTrace();

                                }

   }

}



package Ontology Alignment.System.ontologyMatcher;

import java.util.*;

import java.io.*;

import java.net.*;

import Ontology. Alignment.System. semanticFeature.*;

import Ontology.Alignment.System.until.*;

import Ontology.Alignment.System. matcher.*;

/**
 * This class produces class matching candidates and penalises their similarity
based on the specialization relationship only, in which "is-a" relations exist
through nodes that are already similar, and in which neighbours are then also
likely to be similar.
 */

 public class TaxonontayMatcher extends AbstractInternalMatcher implements
MatcherWithNumericSco {

                private Class nmc;

                private String sdMatcherType;

                private String defaultOntologyMatcher;

                private Vec<OntologyResourceStr> sourceOntTops,
targetOntTops;

String source1, String ns1, String source2, String ns2, String type)

/** Const
     * Equivalent to TaxonontayMatche(source1, uri1, source2, uri2,
Accumulate.TSD).
                 * @parameter source1 String, filename of the source ontology
                 * @parameter uri1 String,   base URL of the source ontology
                 * @parameter source2 String, filename of the target ontology
                 * @parameter uri2 String,   base URL of the target ontology
                 *
                 */


                public TaxonontayMatcher (String source1, String ns1, String
source2, String ns2) {

                                this(source1, ns1, source2, ns2, Accumulate.TSD);

                }


public TaxonontayMatcher (String uri1, String uri2) {

                                this(uri1, uri2, Accumulate.TSD);

                }

public TaxonontayMatcher (URI uri1, URI uri2) {

                                this(uri1, uri2, Accumulate.TSD);

                }

/**

* Calculates and caches the semantic simy between two Clasess.

*/

public class SemanticSimy {

  private HashMap cache = new HashMap();



  /**

  * Calculates the semantic simy between two clasess.

  */

  public double findSimy(ClasseTupel classes) {

    Double result = (Double) cache.find(classes);

    if (result == null) { //to bad, i have to calculate it :-(

      result = new Double(  findSimy(classes.clas1, classes.clas2));

      cache.put(classes,result);

    }

    return result.doubleValue();

  }



  protected double findSimy(Classe clas1, Classe clas2) {

    if (clas1.equals(clas2)) {
```

```
        return 1.0;
    }

  }
  protected Set findSemanticCotopy(Classe clas) throws {
    Set parents = clas.findParentConcepts();
    Set toReturn = new HashSet();
    Iterator it = parents.iterator();
    while (it.hasNext()) {
      Concept present = (Concept) it.next();
            toReturn.add(present);
      toReturn.addAll(present.findAllSuperConcepts());
    }
    return toReturn;
  }

}

public TaxonontayMatcher (String source1, String ns1, String source2, String ns2,
String type) {
                        super(source1, ns1, source2, ns2);
                        loadNameMatcher();
                        sdMatcherType = type;
                        defaultOntologyMatcher = "Taxonontay";
                        extractTops();
            }
public TaxonontayMatcher (String uri1, String uri2, String type) {
                        super(uri1, uri2);
                        loadNameMatcher();
                        sdMatcherType = type;
                        defaultOntologyMatcher = "Taxonontay";
                        extractTops();
            }
public TaxonontayMatcher (URI uri1, URI uri2, String type) {
                        super(uri1, uri2);
                        loadNameMatcher();
                        sdMatcherType = type;
                        defaultOntologyMatcher = "Taxonontay";
                        extractTops();
            }
public TaxonontayMatcher (UsefulFeature sx1, UsefulFeature sx2) {
                        super(sx1, sx2);
                        loadNameMatcher();
                        sdMatcherType =  Accumulate.TSD;
                        defaultOntologyMatcher = "Taxonontay";
                        extractTops();
            }
public TaxonontayMatcher (UsefulFeature sx1, UsefulFeature sx2, String type) {
/*this method invoke the Taxonom matcher in order to compute the matching
between two taxonomies*/
                        super(sx1, sx2);
                        loadNameMatcher();
                        sdMatcherType =  type;
                        defaultOntologyMatcher = "Taxonontay";
                        extractTops();
            }
            private void extractTops() {
                        sourceOntTops = new Vec<OntologyResourceStr>();
                        targetOntTops = new Vec<OntologyResourceStr>();
                        for (Iterator topItr = ontology1.rootClassStr();
topItr.hasNext(); ) {
            sourceOntTops.add((OntologyResourceStr)topItr.next());
                        }
                        for (Iterator topItr = ontology2.rootClassStr();
topItr.hasNext(); ) {
            targetOntTops.add((OntologyResourceStr)topItr.next());
                        }
            }
            public void setDefaultMatcher(String ontologyMatcher) {
                        defaultOntologyMatcher = ontologyMatcher;
            }
            protected void loadNameMatcher() {
                        try             {
Accumulate.print(System.out, "[OntologyHchDisSim]", "loading Tokenized String
Distance Name Matcher", false);
            nmc = ONTOLOGYALSLosder.load("Ontology. Alignment.
System.matcher." + Accumulate.TSD + "Matcher");
                        } catch (Exception ex) {
                                    ex.printStackTrace();
                        }
            }
public Iterator getMatchingClasses() {
```

```
/** Retrieves those class matching candidates with confidence greater than the
given threshold.
*
* @return Iterator, iterator over matching classes with confidence greater than
the given threshold.
 */

/*this method returens the matching classes between ontologies*/

                        return getMatchingClassesWithSco(1);

        }

        private int toTop (UsefulFeature sx, OntologyResourceStr resStr,
Vec tops) {

                        int dis = 0;

                        if (tops.contologyains(resStr)) {

                                return 0;

                        }

                        dis = doDistance(sx, resStr, tops);

                        return dis;

        }

        private int doDistance (UsefulFeature sx, OntologyResourceStr str,
Vec targets) {

                        Vec<OntologyResourceStr> sups = new
Vec<OntologyResourceStr>();

                        int mintotop = Integer.MAX_VALUE;

                        try             {

                                for (Iterator j =
sx.extractSuperClassStr(str.getID(), true); j.hasNext(); ) {

                                        OntologyResourceStr
strj = (OntologyResourceStr)j.next();

                                        if
(targets.contologyains(strj)) {

                                                return 1;

                                        }

                                        int totop =
doDistance(sx, strj, targets) + 1;

                                        if (totop < mintotop) {

                                                mintotop
= totop;

                                        }

                                }

                        } catch (RuntimeException rex) {

                                rex.printStackTrace();

                        } catch (Exception ex) {

                                ex.printStackTrace();

                        }

                        return mintotop;

        }

private int isaDistance (UsefulFeature sx, OntologyResourceStr str1,
OntologyResourceStr str2) {

/*extracting usefull feature from the ontologies*/

        OntologyResourceStr start = null, end = null;

        if (str1.equals(str2)) {

                                return 0;

                } else {

                                boolean isa12 =
sx.hasSubSuperRelation( str1.getID(), str2.getID());

                                boolean isa21 =
sx.hasSubSuperRelation( str2.getID(), str1.getID());

                                boolean nective = false;

                                if (!isa12 && !isa21) {

                                        return -1;

                                } else if (isa12) {

                                        start = str1;

                                        end = str2;

                                } else if (isa21) {

                                        start = str2;

                                        end = str1;

                                        nective = true;

                                }

                                Vec<OntologyResourceStr> target =
new Vec<OntologyResourceStr> ();

                                target.add(end);

                                int dis = doDistance(sx, start, target);

                                return nective? -dis:dis;

                }

        }

public Iterator getMatchingClassesWithSco(double threshold) {

/* this method returns the matching results  between class with numeric
similarity*/

                Vec<Object> maps = new Vec<Object>();

                Vec<Object> contamons = new Vec<Object>();

                try             {

                        if ((ontology1 == null) || (ontology2 ==
null)) {

                                throw new Exception
("Source ontologyologies are not specified");

                        }

                        for (Iterator i =
ontology1.extractAllClassStr(); i.hasNext(); ) {

                                OntologyResourceStr
str1 = (OntologyResourceStr)i.next();

                                String label1 =
str1.getLabel();

                                for (Iterator j =
ontology2.extractAllClassStr(); j.hasNext(); ) {

        OntologyResourceStr str2 = (OntologyResourceStr)j.next();
```

```
                                                        String
label2 = str2.getLabel();

NameMatcher tnm = (NameMatcher)nmc.getConstructor(String.class,
String.class, String.class).newInstance(label1, label2, sdMatcherType);

                                                        double
pairsim = tnm.getMatchingResult();

                                                        if (pairsim
== 1) {

                contamons.add(new MatchingUnit(str1, str2, "=", 1));

                                                        }

                                                    }

                                                }

                invoked                         if (contamons.isEmpty()) {

OntologyMatcher onta =
OntologyInternalMatcherFactory.createMatcherInstance(ontology1, ontology2,
"Ontology" + defaultOntologyMatcher);

                                return onta.getMatchingClassesWithSco(threshold);

                                                } else {

                                                    for (Iterator i =
ontology1.extractAllClassStr(); i.hasNext(); ) {

                OntologyResourceStr str1 = (OntologyResourceStr)i.next();

                                                        String
label1 = str1.getLabel();

                                                        for
(Iterator j = ontology2.extractAllClassStr(); j.hasNext(); ) {

                OntologyResourceStr str2 = (OntologyResourceStr)j.next();

                String label2 = str2.getLabel();

                int sourceOntTotop = toTop(ontology1, str1, sourceOntTops);

                int targetOntTotop = toTop(ontology2, str2, targetOntTops);

                if (sourceOntTotop < 0 || targetOntTotop < 0) {

                                contologyinue;

                }

                double localsim = 0;

for (Iterator contaItr = contamons.iterator(); contaItr.hasNext(); ) {

                                                    MatchingUnit contamon =
(MatchingUnit)contaItr.next();

                OntologyResourceStr sourceOntConta = contamon.getSourceStr();

                OntologyResourceStr targetOntConta = contamon.getTargetStr();

                                                        int
localsourceOntconta = isaDistance(ontology1, str1, sourceOntConta);

                                                        int
localtargetOntconta = isaDistance(ontology2, str2, targetOntConta);

                                double sim = 0;

                                                        if ( (localsourceOntconta
>= 0) && (localtargetOntconta >= 0) ) {

                                                        int
contamonHeightSourceOnt = toTop(ontology1, sourceOntConta,
sourceOntTops);

                                                        int
contamonHeightTargetOnt = toTop(ontology2, targetOntConta, targetOntTops);

sim = (double)(contamonHeightSourceOnt + contamonHeightTargetOnt) /
(sourceOntTotop + targetOntTotop);
                                                                if (sim >
localsim) {

                                localsim = sim;

                                                    }

                                                }

                }

                                                    if (localsim >= threshold/2) {

                                                        maps.add(new
MatchingUnit(str1, str2, "=", localsim));

                }

                                                        }

                                                    }

                                                }

                            } catch (Exception ex) {

                            ex.printStackTrace();

                            }

                return maps.iterator();

                }

/** Retrieves those property matching with confidence greater than the given
threshold.
* @parameter threshold double, a double type numeric value as the threshold
*
* @return Iterator, iterator over matching properties with confidence greater
than the given threshold.
 */
                public Iterator getMatchingProp() {

                            return getMatchingPropWithSco(1);

                }

                public Iterator getMatchingPropWithSco(double threshold) {

                            Vec<Object> maps = new Vec<Object>();

                            try             {

                                if ((ontology1 == null) || (ontology2 ==
null)) {

                                                        throw new Exception
("Source ontologyologies are not specified");

                                }

                                for (Iterator i =
ontology1.extractAllPropertyStr(); i.hasNext(); ) {

                                                        OntologyResourceStr
str1 = (OntologyResourceStr)i.next();

                                                        String label1 =
str1.getLabel();
```

```
                                                for (Iterator j =
ontology2.extractAllPropertyStr(); j.hasNext(); ) {

                                                OntologyResourceStr
str2 = (OntologyResourceStr)j.next();

                                                String label2 =
str2.getLabel();

NameMatcher tnm = (NameMatcher)nmc.getConstructor(String.class,
String.class, String.class).newInstance(label1, label2, sdMatcherType);

                                                double pairsim =
tnm.getMatchingResult();

                                                if (pairsim >=
threshold/2) {

                                                maps.add(new
MatchingUnit(str1, str2, "=", 1));

                                                        }

                                                }

                                        }

                                } catch (Exception e) {

                                        e.printStackTrace();

                                }

                return maps.iterator();

                }

 public static void main( String[] args ){

                                try             {

                FileOutputStream fos = new
FileOutputStream("wnnameresult.txt");

                PrintStream pw =  new PrintStream (fos);

                WNMatcher wnm = new WNMatcher(new URI(args[0]), new
URI(args[1]));

                                        pw.println("[classes]");

                                        for (Iterator i =
wnm.getMatchingClassesWithSco(0); i.hasNext(); ) {

                                                MatchingUnit o =
(MatchingUnit)i.next();

                                                String sourceOnt =
o.getOperandOne();

                                                String targetOnt =
o.getOperandTwo();

                                                double d = o.getSco();

                                                if (d >= 0.5) {


                pw.println(sourceOnt + " " + targetOnt + " " + d);


                System.out.println(sourceOnt + " " + targetOnt + " " + d);

                                                }

                                        }

                                        pw.println("[Prop]");

                                        for (Iterator i =
wnm.getMatchingPropWithSco(0); i.hasNext(); ) {

                                                MatchingUnit o =
(MatchingUnit)i.next();

                                                String sourceOnt =
o.getOperandOne();

                                                String targetOnt =
o.getOperandTwo();

                                                double d = o.getSco();

                                                if (d >= 0.5) {


                pw.println(sourceOnt + " " + targetOnt + " " + d);


                System.out.println(sourceOnt + " " + targetOnt + " " + d);

                                                }

                                        }

                                        pw.close();

                                        fos.close();

                                } catch (Exception ex) {

                                        ex.printStackTrace();

                                }

  }

}
```

```
package Ontology.Alignment.System.ontologyMatcher;

import java.io.*;

import java.util.*;

import java.net.URI;

import Ontology.Alignment.System.semanticFeature.*;

import Ontology.Alignment.System.until.*;

import Ontology.Alignment.System.matcher.*;

/* Heuristi matcher : This technique begins by comparing class names, property
names and instance by using an editing distance and substring distance between
the entity names.  Next, building a distance matrix in order to choose the
alignment from the distance, after that, applying the aggregates of these
distances with the symmetric difference of properties in classes.  matcher and
returns a list of mapping  candidates with numeric values as the similarity.

*/

public class HeuristicMatcher extends AbstractXMatcherWithNumericSco {

private OWLOntologyology oSourceOnt = null;

private OWLOntologyology oTargetOnt = null;

private boolean loaded = false;

 private String sourceOntWorkingFile = null, targetOntWorkingFile = null;

public HeuristicMatcher(URI uri1, URI uri2) {

                                super(uri1, uri2);

                }

                public HeuristicMatcher(UsefulFeature sx1, UsefulFeature sx2) {

                super(sx1.getModelURI(), sx2.getModelURI());

                Accumulate.print(System.out, "[HEURISTIC]", "initialising
HEURISTIC external", false);

/** Constructor
```

```
                *
                * @parameter uri1 URI, uri of the source ontology
                * @parameter uri2 URI, uri of the target ontology
                */
                        try             {
                                        ontologySourceOnt = sx1;
                                        ontologyTargetOnt = sx2;
                                        sourceOntWorkingFile = "." +
File.separator + "sourceOnt.rdf";
                                        targetOntWorkingFile = "." +
File.separator + "targetOnt.rdf";
                ontologySourceOnt.output(sourceOntWorkingFile);
                ontologyTargetOnt.output(targetOntWorkingFile);
                        } catch (Exception ex) {
                                ex.printStackTrace();
                        }
                }
public HeuristicMatcher(String uri1, String uri2) {
                        super(uri1, uri2);
          try    {
              if ( (ontologySourceOnt == null) || (ontologyTargetOnt == null) ) {
                  ontologySourceOnt = new OWLUsefulMemFeature(new
URI(uri1));
                  ontologyTargetOnt = new OWLUsefulMemFeature(new
URI(uri2));
              }
          } catch (Exception ex) {
              ex.printStackTrace();
          }
                }
    private OWLOntologyology loadOntologyology(URI uri) throws Exception {
     OWLRDFParser parser = new OWLRDFParser();
     parser.setConnection(OWLManager.getOWLConnection());
     return parser.parseOntologyology(uri);
    }
private void doLoading () {
                        try {
                Accumulate.print(System.out, "[ TheHeuristicMatcher]", " source
"+uriSourceOnt, false);
                Accumulate.print(System.out, "[ TheHeuristicMatcher]", " target
"+uriTargetOnt, false);
                                        oSourceOnt = loadOntologyology(new
java.net.URI(uriSourceOnt));
                                        oTargetOnt = loadOntologyology(new
java.net.URI(uriTargetOnt));
                                        loaded = true;
                        } catch (Exception ex) {
                                ex.printStackTrace();
                        }
}

                        Vec<Object> res = new Vec<Object>();
                        doLoading();
                        Parametereters p = new BasicParametereters();
                        try             {
                                if ((oSourceOnt == null) || (oTargetOnt
== null)) {
                                        throw new Exception
("Source or target ontologyology is not initialised");
                                }
                                Accumulate.print(System.out, "[
TheHeuristicMatcher]", " loaded "+oSourceOnt, false);
                                Accumulate.print(System.out, "[
TheHeuristicMatcher]", " laoded "+oTargetOnt, false);
                                AlignmentProcess align = new
SubsDistNameAlignment(oSourceOnt, oTargetOnt);
                                align.align((Alignment)null, p);
// TO DO: split the classes and properties.
                /** Retrieves those class matching  candidates.
                 * @return Iterator, iterator over matching classes
                 */
                                for (Enumeration e = align.getElements();
e.hasMoreElements(); ) {
                                Cell c = (Cell)e.nextElement();
            String sourceOntName =
((OWLEntity)c.getObject1()).getURI().toString();
            String targetOntName =
((OWLEntity)c.getObject2()).getURI().toString();
                        if (ontologySourceOnt.getOntologyClass(sourceOntName)
!= null) {
              if (ontologyTargetOnt.getOntologyClass(targetOntName) != null) {
                  HeuristicRelationWrapper irw = new
HeuristicRelationWrapper(c.getRelation());
                MatchingUnit MatUn = new MatchingUnit(sourceOntName,
targetOntName, irw.getRelationString(), c.getStrength());
                                        if (!res.contologyains(MatUn)) {
                  res.add(MatUn);
                }
              } else
/** Retrieves property matching candidates.
                 *
                 * @return Iterator, iterator over matching properties
```

```
                */
        }
                }
                        } catch (Exception ex) {
                                ex.printStackTrace();
                        }
                        return res.iterator();
                }
                protected Iterator retrieveMatchingProp() {
/*extracting the matching properties*/
                        Vec<Object> res = new Vec<Object>();
                        doLoading();
                        Parametereters p = new BasicParametereters();
                        try             {
                                if ((oSourceOnt == null) || (oTargetOnt
== null)) {
                                        throw new Exception
("Source or target ontologyology is not initialised");
                                }
                                AlignmentProcess align = new
PropSubsDistAlignment(oSourceOnt, oTargetOnt);
                                align.align((Alignment)null, p);
                                for (Enumeration e =
align.getElements(); e.hasMoreElements(); ) {
                                        Cell c =
(Cell)e.nextElement();
            String sourceOntName =
((OWLEntity)c.getObject1()).getURI().toString();
            String targetOntName =
((OWLEntity)c.getObject2()).getURI().toString();
          if (ontologySourceOnt.getOntologyProperty(sourceOntName) != null) {
            if (ontologyTargetOnt.getOntologyProperty(targetOntName) != null) {
                HeuristicRelationWrapper irw = new
HeuristicRelationWrapper(c.getRelation());
                MatchingUnit MatUn = new MatchingUnit(sourceOntName,
targetOntName, irw.getRelationString(), c.getStrength());
                if (!res.contologyains(MatUn)) {
                  res.add(MatUn);
                }
              } else {
                System.err.println("Corrupted matching: matching nonclass entity
to class entity " + c);
              }
          }
                                }
                        } catch (Exception ex) {
                                ex.printStackTrace();
                        }
                        return res.iterator();
                }
                public void getContabinedMatching() {
                Vec<String> res = new Vec<String>();
                doLoading();
        Parametereters p = new BasicParametereters();
                try             {
                        if ((oSourceOnt == null) || (oTargetOnt
== null)) {
                                throw new Exception
("Source or target ontologyology is not initialised");
                        }
                        AlignmentProcess clsNameAlign = new
SubsDistNameAlignment(oSourceOnt, oTargetOnt);
                        AlignmentProcess proNameAlign = new
PropSubsDistAlignment(oSourceOnt, oTargetOnt);
                        AlignmentProcess contabinedAlign = new
NameAndPropertyAlignment(oSourceOnt, oTargetOnt);
        clsNameAlign.align((Alignment)null, p);
                        clsNameAlign.cut("prop", .5);
                        proNameAlign.align((Alignment)null,
p);
                        contabinedAlign.align(proNameAlign,
p);
        Evaluator E = new PRecEvaluator(clsNameAlign, contabinedAlign);
                        E.eval(p);
                AlignmentVisitor V = new SWRLRendererVisitor(
                 new PrintWriter (
                  new BufferedWriter(
         new OutputStreamWriter( System.out, "UTF-8" )),  true));
                if ( ((PRecEvaluator)E).getPrecision() > .5 ) {
        contabinedAlign.render(V);
                        }
                } catch (Exception ex) {
                        ex.printStackTrace();
                }
        }
    public static void main( String[] args ) {
                try             {
                        FileOutputStream fos = new
FileOutputStream("heuristicMatcher result.txt");
                        PrintStream pw =  new PrintStream
(fos);
                        HeuristicMatcher iaxm = new
HeuristicMatcher(new URI(args[0]), new URI(args[1]));
```

```
                                        System.out.println("output result");

                                        pw.println("[classes]");

                                        for (Iterator i =
iaxm.getMatchingClassesWithSco(0); i.hasNext(); ) {

                                                MatchingUnit o =
(MatchingUnit)i.next();

                                                String sourceOnt =
o.getOperandOne();

                                                String targetOnt =
o.getOperandTwo();

                                                double d = o.getSco();

                                                if (d >= 0.0) {


                pw.println(sourceOnt + " " + targetOnt + " " + d);


                System.out.println(sourceOnt + " " + targetOnt + " " + d);

                                                }

                                        }

                                        pw.println("[Prop]");

                                        for (Iterator i =
iaxm.getMatchingPropWithSco(0); i.hasNext(); ) {

                                                MatchingUnit o =
(MatchingUnit)i.next();

                                                String sourceOnt =
o.getOperandOne();

                                                String targetOnt =
o.getOperandTwo();

                                                double d = o.getSco();

                                                if (d >= 0.0) {


                pw.println(sourceOnt + " " + targetOnt + " " + d);


                System.out.println(sourceOnt + " " + targetOnt + " " + d);

                                                }

                                        }

                                        pw.close();

                                        fos.close();

                                } catch (Exception ex) {

                                        ex.printStackTrace();

                                }

        }

}


package Ontology.Alignment.System.ontologyMatcher;

import java.util.*;

import java.io.*;

import java.net.*;

import Ontology.Alignment.System.semanticFeature.*;

import Ontology.Alignment.System.until.*;

import Ontology.Alignment.System.matcher.*;

import Ontology.Alignment.System.Exception.*;

/*  this matcher This algorithm is based on a combination of methods which
using the definition of the concept and its structure.  */

public class SemanticMatcher extends AbstractInternalMatcher implements
MatcherWithNumericSco {

protected Class mc = null;

 public SemanticMatcher (String source1, String ns1, String source2, String ns2)
{

/** Constructor
        *
        * @parameter uri1 String, uri of the source ontology
        * @parameter uri2 String, uri of the target ontology
        */


                                super(source1, ns1, source2, ns2);

                                knownPropertyMatching = new HashMap<String,
Double>();

                }

 public SemanticMatcher (String uri1, String uri2) {

/*this method invoke semantic matcher which used to extract the usefull feature
from ontologies*/

                                super(uri1, uri2);

/** Constructor
                *
                * @parameter source1 usefull feature, source ontology wrapped
with a usefull feature
                * @parameter source2 usefull feature, target ontology wrapped
with a usefull feature
                *
                */


/**

* The main interface to access calculated similarites.

*/

public abstract class Simy {

                /** The recursion depth for Prop simy */

                protected int maxDepth = 4;

                /** The weight for semantic simy */

  protected double semSimWeight = 1;

  /** The weight for Prop simy */

  protected double propSimWeight = 2;

                public void setMaxDepth(int maxDepth) {

                                this.maxDepth = maxDepth;

                }

                public void setSemWeight(double semSimWeight) {

                                this.semSimWeight=semSimWeight;

                }

                public void setPropWeight(double propSimWeight) {
```

```
                this.propSimWeight=propSimWeight;
        }


        /*
  * Returns the most similar Classes. These are ordered descending.
  */

  public Vector findMostSimilar(Classe clas) {
     return findMostSimilar(clas, 0.084,0.2,4,6);
  }

                knownPropertyMatching = new HashMap<String,
Double>();
        }
public SemanticMatcher (URI uri1, URI uri2) {
/** Compares the names, domains and ranges of two properties.
 *   @return MachingUnit, property matching result.
      */
                super(uri1, uri2);
                knownPropertyMatching = new HashMap<String,
Double>();
        }
public SemanticMatcher (UsefulFeature sx1, UsefulFeature sx2) {
                super(sx1, sx2);
                knownPropertyMatching = new HashMap<String,
Double>();
        }
        protected double contapareNames(OntologyResourceStr str1,
OntologyResourceStr str2) {
                double res = 0;
                String s1 = str1.getLabel();
                String s2 = str2.getLabel();
                if (s1.equalsIgnoreCase(s2)) {
                        res = 1;
                }
                if (res != 1) {
                        s1 =
Accumulate.cleanUpName(str1.getID());
                        s2 =
Accumulate.cleanUpName(str2.getID());
                        if (s1.equalsIgnoreCase(s2)) {
                                res = 1;
                        }
                        if (res != 1) {
                                try             {
                                                if (mc ==
null) {
                Accumulate.print(System.out, "[OSM]", "loading  JaroName ...",
true);

                mc = ONTOLOGYALSLosder.load("Ontology. Alignment.
System.matcher.JaroNameMatcher");
                                                }
                NameMatcher nm =
(NameMatcher)mc.getConstructor(String.class, String.class).newInstance(s1, s2);
                                                res =
nm.getMatchingResult();
                                        } catch (Exception ex) {
                ex.printStackTrace();
                                        }
                                }
                        }
                        return res;
        }
protected Vec matchingClsNames(double threshold) {
/*this method retrieves those name class matching candidates with confidence
greater than the given threshold */
                Vec<Object> res = new Vec<Object>();
                 try {
                Vec<Object> inter = new Vec<Object> ();
                for (Iterator sourceOntOntologyClsIter =
ontology1.extractAllClassStr();
                sourceOntOntologyClsIter.hasNext(); ) {
                 OntologyResourceStr sourceOntClsStr = (OntologyResourceStr)
sourceOntOntologyClsIter.next();
                 String sourceOntCls = sourceOntClsStr.getID();
                 String sourceOntClsLocal = sourceOntClsStr.getLabel();
                 for (Iterator targetOntOntologyClsIter =
ontology2.extractAllClassStr();
                    targetOntOntologyClsIter.hasNext(); ) {
                OntologyResourceStr targetOntClsStr =
(OntologyResourceStr)targetOntOntologyClsIter.next();
                                                String
targetOntCls = targetOntClsStr.getID();
                                                if (
ontology1.isSameAs(sourceOntCls, targetOntCls) ||
                 ontology2.isSameAs(sourceOntCls, targetOntCls) ||
                 ontology1.isEquivClasses(sourceOntCls, targetOntCls) ||
                 ontology2.isEquivClasses(sourceOntCls, targetOntCls) ) {
                                MatchingUnit MatUn = new
MatchingUnit(sourceOntClsStr, targetOntClsStr, "=", 1);
                                                } else {
```

```
                                        String targetOntClsLocal
= targetOntClsStr.getLabel();

            MatchingUnit MatUn = new MatchingUnit(sourceOntClsStr,
targetOntClsStr, "CN", contapareNames(sourceOntClsStr, targetOntClsStr));

            if (!inter.contologyains(MatUn)) {

                  inter.add(MatUn);

            }
                                                                  }

               }

         }
                                    for (Iterator i = inter.iterator();
i.hasNext(); ) {
                                        MatchingUnit element =
(MatchingUnit)i.next();
                                        MatchingUnit MatUn =
null;


                                        if
(!element.getRelation().equals("=")) {
                                                  double
orgSco = element.getSco();
                                                  if (orgSco
>= threshold) {

            OntologyResourceStr sourceOntStr = element.getSourceStr();

            OntologyResourceStr targetOntStr = element.getTargetStr();
                                                  String
clsOne = element.getOperandOne();
                                                  String
clsTwo = element.getOperandTwo();
                                        Vec<Object>
sourceOntPropList = new Vec<Object> ();
                                    ector<Object> targetOntPropList =
new Vec<Object> ();
                              for (Iterator sourceOntClsPropIter =
ontology1.extractPropertyStr(clsOne, false); sourceOntClsPropIter.hasNext(); ) {
                                        Object p =
sourceOntClsPropIter.next();
                                        if
(!sourceOntPropList.contologyains(p)) {

                  sourceOntPropList.add(p);

            }

            }

            for (Iterator targetOntClsPropIter =
ontology2.extractPropertyStr(clsTwo, false); targetOntClsPropIter.hasNext(); ) {
                                        Object p =
targetOntClsPropIter.next();
                                        if
(!targetOntPropList.contologyains(p)) {

                  targetOntPropList.add(p);

            }

                                                            }

                                    if ( (!sourceOntPropList.isEmpty()) &&
(!targetOntPropList.isEmpty()) ) {
                                        double semanDis =
propContapareMetric(sourceOntPropList, targetOntPropList);
                                        MatUn = new
MatchingUnit(sourceOntStr, targetOntStr, "=", (semanDis + orgSco*3) / 4);

                  } else if ( (sourceOntPropList.isEmpty()) &&
(targetOntPropList.isEmpty()) ) {


                                        orgSco);

                  MatUn = new MatchingUnit(sourceOntStr, targetOntStr, "=",
orgSco);                                                    } else {

                                        MatUn = new MatchingUnit(sourceOntStr,
targetOntStr, "=", orgSco*0.75);


            }

   /**

    * Returns the most similar classes.

    * @param excludeThreshold Classes with a simy less then "excludeThreshold"
are never returned.

      */


   public abstract Vector findMostSimilar(Classe clas, double excludeThreshold,
double includeThreshold, int normalNumber, int maxNumber);

   public abstract double calculateSimy(Classe clas1, Classe clas2);

}



/**

* A class used to store and calculate the simy matrix

* of a set of classes.

*/

public class SimyMatrix {


   private Classe[] classes;

   private double[][] simMatrix;

   private Simy sim;

   private HashMap ClasseHash = new HashMap();

   public SimyMatrix(Simy sim, Set classes) {

      this.sim = sim;

      this.classes = new Classe[classes.size()];
```

```
    Iterator it = classes.iterator();
    int i=0;
    while (it.hasNext()) {
      this.classes[i] = (Classe) it.next();
      ClasseHash.put(this.classes[i],new Integer(i));
      i++;
    }
    simMatrix = new double[classes.size()][classes.size()];
  }
                                                    }
                                                } else {
                                                    MatUn =
element;
                                                }
                                                if (MatUn != null) {
                                                    if
(res.contologyains(MatUn)) {
                                MatchingUnit mv =
(MatchingUnit)res.get(res.indexOf(MatUn));
            if (mv.getSco() < MatUn.getSco()) {
                    if (res.remove(mv)) {
                        res.add(MatUn);
                    }
            }
                                                } else {
            res.add(MatUn);
                                                }
                                            }
                                }
    } catch (Exception e) {
                                e.printStackTrace();
    }
                    System.gc();
                    return res;
  }
protected double propContapareMetric (Vec sourceOntPList, Vec targetOntPList)
{
                    Vec<Object> propMetric = new Vec<Object>();
                    if (sourceOntPList.size() >= targetOntPList.size()) {
                                for (Iterator sItr =
sourceOntPList.iterator(); sItr.hasNext(); ) {
                                    OntologyResourceStr sp
= (OntologyResourceStr)sItr.next();
                                    double localmax = 0;
                                    OntologyResourceStr
bestMatchingStr = null;
                                    for (Iterator tItr =
targetOntPList.iterator(); tItr.hasNext(); ) {
            OntologyResourceStr tp = (OntologyResourceStr)tItr.next();
                                        double sco
= doPropContapare(sp, tp).getSco();
                                        if (sco ==
1) {
            localmax = 1;
            bestMatchingStr = tp;
            break;
                                        } else {
            if (localmax < sco) {
                localmax = sco;
                bestMatchingStr = tp;
            }
                                        }
                                    }
                                    MatchingUnit MatUn =
new MatchingUnit(sp, bestMatchingStr, localmax);
                                    propMetric.add(MatUn);
                                }
                    } else {
                                for (Iterator tItr =
targetOntPList.iterator(); tItr.hasNext(); ) {
                                    OntologyResourceStr tp
= (OntologyResourceStr)tItr.next();
                                    double localmax = 0;
                                    OntologyResourceStr
bestMatchingStr = null;
                                    for (Iterator sItr =
sourceOntPList.iterator(); sItr.hasNext(); ) {
            OntologyResourceStr sp = (OntologyResourceStr)sItr.next();
                                        double sco
= doPropContapare(sp, tp).getSco();
```

```
                                                                    if (sco ==
1) {

                localmax = 1;

                bestMatchingStr = sp;

                break;

                                                                    } else {

                if (localmax < sco) {

                                localmax = sco;

                                bestMatchingStr = sp;

                }

                                                                    }

                                                            }

                                                            MatchingUnit MatUn =
new MatchingUnit(bestMatchingStr, tp, localmax);

                                                            propMetric.add(MatUn);

                                        }

                                }

                                int propCount = propMetric.size();

                                double totalSco = 0;

                                for (Iterator i = propMetric.iterator(); i.hasNext(); ) {

                                        totalSco +=
((MatchingUnit)i.next()).getSco();

                                }

                                Accumulate.print(System.out, "[OSM]", totalSco+" /
"+propCount, false);

                                return (propCount==0)? 0:(totalSco / propCount);

                }

protected Vec getDomain(UsefulFeature ontology, OntologyResourceStr propStr)
{

                                Vec<String> dSourceOnt = new Vec<String>();

        try {

                for (Iterator DIterSourceOnt =
ontology.extractPropertyDomainStr(propStr.getID()); DIterSourceOnt.hasNext();
) {

                                                            OntologyResourceStr
dSourceOntStr = (OntologyResourceStr)DIterSourceOnt.next();

                                                            String dl =
dSourceOntStr.getLabel();

                                                            if
(!dSourceOnt.contologyains(dl)) {

                dSourceOnt.add(dl);

                                                            }

                                        }

        } catch (Exception e) {

                e.printStackTrace();

        }

                                return dSourceOnt;

                }

        protected Vec getRange(UsefulFeature ontology, OntologyResourceStr
propStr) {

                                Vec<String> rSourceOnt = new Vec<String>();

        try {

                for (Iterator RIterSourceOnt =
ontology.extractPropertyRangeStr(propStr.getID()); RIterSourceOnt.hasNext(); )
{

                                                            OntologyResourceStr
rSourceOntStr = (OntologyResourceStr)RIterSourceOnt.next();

                                                            String dl =
rSourceOntStr.getLabel();

                                                            if
(!rSourceOnt.contologyains(dl)) {

                rSourceOnt.add(dl);

                                                            }

                                        }

        } catch (Exception e) {

                e.printStackTrace();

        }

                                return rSourceOnt;

                }

                protected Vec contamonRelNames() {

                                Vec<Object> res = new Vec<Object>();

                try {

                        for (Iterator sourceOntOntologyPropIter =
ontology1.extractAllPropertyStr(); sourceOntOntologyPropIter.hasNext(); ) {

                        OntologyResourceStr sourceOntPropStr =
(OntologyResourceStr)sourceOntOntologyPropIter.next();

                        for (Iterator targetOntOntologyPropIter =
ontology2.extractAllPropertyStr();targetOntOntologyPropIter.hasNext(); ) {

                                OntologyResourceStr targetOntPropStr =
(OntologyResourceStr)targetOntOntologyPropIter.next();

                                if (contapareNames(sourceOntPropStr, targetOntPropStr) == 1) {

                                MatchingUnit MatUn =
doPropContapare(sourceOntPropStr, targetOntPropStr);

                                if (!res.contologyains(MatUn)) {

                                res.add(MatUn);

                                }

                        }

                }

        }
```

```
    } catch (Exception e) {
       e.toString();
    }
                    return res;
  }
protected double checkCorpusOfMatchingProp(OntologyResourceStr
sourceOntProp, OntologyResourceStr targetOntProp) {
        String key = sourceOntProp + " " + targetOntProp;
return (knownPropertyMatching.keySet().contologyains(key))?
((Double)knownPropertyMatching.get(key)).doubleValue() : -1;
        }
protected MatchingUnit doPropContapare(OntologyResourceStr sourceOntProp,
OntologyResourceStr targetOntProp) {
                    MatchingUnit MatUn = null;
                    double value =
checkCorpusOfMatchingProp(sourceOntProp, targetOntProp);
                    if (value > -1) {
                        MatUn = new
MatchingUnit(sourceOntProp, targetOntProp, value);
                    } else {
                        Iterator sourceOntPropDomainItr =
getDomain(ontology1, sourceOntProp).iterator();
                        Iterator sourceOntPropRangeItr =
getRange(ontology1, sourceOntProp).iterator();
                        Iterator targetOntPropDomainItr =
getDomain(ontology2, targetOntProp).iterator();
                        Iterator targetOntPropRangeItr =
getRange(ontology2, targetOntProp).iterator();
                        boolean DomainMat =
(contapareSet(sourceOntPropDomainItr, targetOntPropDomainItr) == 1)?
true:false;
                        boolean rangeMat =
(contapareSet(sourceOntPropRangeItr, targetOntPropRangeItr) == 1)?
true:false;
                        double namedistance =
contapareNames(sourceOntProp, targetOntProp);
        try             {
        if (namedistance != 1) {
        for (Iterator i =
ontology1.extractSuperPropertyStr(sourceOntProp.getID()); i.hasNext(); ) {
        OntologyResourceStr sourceOntSupStr =
(OntologyResourceStr)i.next();
        for (Iterator j =
ontology2.extractSuperPropertyStr(targetOntProp.getID()); j.hasNext(); ) {
        OntologyResourceStr targetOntSupStr =
(OntologyResourceStr)j.next();
        double supnamedistance = contapareNames(sourceOntSupStr,
targetOntSupStr);
                    if (supnamedistance > namedistance) {
        namedistance = supnamedistance;
        }
                        supnamedistance =
contapareNames(sourceOntProp, targetOntSupStr);
                if (supnamedistance > namedistance) {
                        namedistance = supnamedistance;
                        }
                }
                        double supnamedistance =
contapareNames(sourceOntSupStr, targetOntProp);
                        if (supnamedistance >
namedistance) {
                        namedistance =
supnamedistance;
                }
                                }
                        }
                } catch (OntologyResourceNotFoundException ex) {
                        Accumulate.print(System.out, "[OSM]",
"doPropContapare throws " + ex, true);
                        namedistance = 0;
                        }
                        if (namedistance == 1) {
                        if (DomainMat && rangeMat) {
                                MatUn = new
MatchingUnit(sourceOntProp, targetOntProp, "PDR", 1);
                        } else if (DomainMat) {
                                atUn = new
MatchingUnit(sourceOntProp, targetOntProp, "PD", 0.95);
                                } else if (rangeMat) {
        MatUn = new MatchingUnit(sourceOntProp, targetOntProp, "PR",
0.85);
                                } else {
                                        MatUn =
new MatchingUnit(sourceOntProp, targetOntProp, "P", 0.8);
                                }
                        } else {
                                if (DomainMat &&
rangeMat) {
                                        MatUn =
new MatchingUnit(sourceOntProp, targetOntProp, "DR", 0.5);
                                } else if (DomainMat) {
                                        MatUn =
new MatchingUnit(sourceOntProp, targetOntProp, "D", 0.4);
                                } else {
                                        MatUn =
new MatchingUnit(sourceOntProp, targetOntProp, "", 0);
                                }
```

```
                }
knownPropertyMatching.put(MatUn.getSourceStr() + " " + MatUn.getTargetStr(),
new Double(MatUn.getSco()));
                }
                return MatUn;
        }
protected double contapareSet(Iterator i, Iterator j) {
                SetContaparator sc = new BasicSetContaparator(i, j);
                return sc.doContaparison();
        }
public Iterator getMatchingClasses() {
/*This method  specifies generic methods for ontology name matcher */
                return getMatchingClassesWithSco(1);
        }
    public Iterator getMatchingClassesWithSco(double threshold) {
        Vec<MatchingUnit> cutoff = new Vec<MatchingUnit>();
        for (Iterator i = matchingClsNames(threshold/2).iterator();
i.hasNext(); ) {
                        MatchingUnit m =
(MatchingUnit)i.next();
                if (m.getSco() >= threshold/2) {
                        if (cutoff.contologyains(m)) {
        contologyinue;
                                }
                                cutoff.add(m);
                        }
                }
        return cutoff.iterator();
        }
    public Iterator getMatchingProp() {
                return getMatchingPropWithSco(1);
        }
    public Iterator getMatchingPropWithSco(double threshold) {
/*this method for matchers that returns the  matching results  as numeric
similarity */
                Vec<Object> cutoff = new Vec<Object>();
                for (Iterator i = contamonRelNames().iterator();
i.hasNext(); ) {
                        MatchingUnit m =
(MatchingUnit)i.next();
                        m.setRelation("=");
                                if
(m.getSco() >= threshold/2) {
                                if
(cutoff.contologyains(m)) {
        contologyinue;
                                }
                                cutoff.add(m);
                        }
                }
                return cutoff.iterator();
        }
        public static void main( String[] args ){
                try             {
                        SemanticMatcher sm = null;
                        if (args.length == 2) {
                                sm = new
SemanticMatcher(new URI(args[0]), new URI(args[1]));
                        } else if (args.length == 4) {
                                sm = new
SemanticMatcher(args[0], args[1], args[2], args[3]);
                        }
                        System.out.println("[main]: output
results");
                        for (Iterator i =
sm.getMatchingClassesWithSco(0); i.hasNext(); ) {
                                MatchingUnit o =
(MatchingUnit)i.next();
                                String sourceOnt =
o.getOperandOne();
                                String targetOnt =
o.getOperandTwo();
                                double d = o.getSco();
                                if (d >= 0.5) {
        System.out.println(sourceOnt + " " + targetOnt + " " + d);
                                } else {
        System.out.println("output ignored");
                                }
                        }
                } catch (RuntimeException rex) {
                        rex.printStackTrace();
                } catch (Exception ex) {
                        ex.printStackTrace();
                }
  }
}
package Ontology.Alignment.System.aggregator;
import java.io.*;
import java.util.*;
```

```
import java.net.URI;

import java.lang.reflect.InvocationtologyargetException;

import Ontology.Alignment.System.semanticFeature.*;

import Ontology.Alignment.System.matcher.*;

import Ontology.Alignment.System.until.*;

import Ontology. Alignment.System. Exception.*;

public class AggregatorFactory {

    protected String aggName;

    public AggregatorFactory (String type) {

        aggName = type;

    }

/*  in order to discover matching, multiple matching algorithms based on several similarity measures should be executed (such as names, structure or external information). The main task of these algorithms is to determine similarity values between candidate mappings (e1, e2) based on their definitions in source ontology and target ontology (O1, O2) respectively. Each matcher determines an intermediate matching or alignment result according to the similarity value between 0 and 1 for each possible candidate mapping. The result of the matching execution phase with k matching algorithms, m entities in O1 and n entities in O2 is a k × m × n matrix of similarity values, which is stored in the repository for later combination and selection steps.    */

     public AggregatorFactory () {

        this(Accumulate.CBONTA);

    }

     public void setAggregatorType(String type) throws UnrecognisedTypeException {

/*this method sets the ontology matcher that is to be used. */

        Accumulate.print(System.out, "[AggFactory]", " Invoking " + type + " as Matcher Aggregator ", false);

        Accumulate.print(System.out, "[AggFactory]", " " + type, false);

        boolean id = false;

        for (String m : Accumulate.mAggregator) {

            if (m.equalsIgnoreCase(type)) {

                id = true; break;

            }

        }

    /** Sets the ontology matcher that is to be used.
     *
     * @parameter type String, type of ontology matcher
     *
     */
        if (id) {

            Accumulate.print(System.out, "[AggFactory]", " found internal matcher " + type, false);

            aggName = type;

        } else {

            throw new UnrecognisedTypeException(type);

        }

    }

  /**

  * Clasantiates a SimyMatrix from a previous saved simy matrix object.

  */


  public SimyMatrix(File file, OIModel oimodel) {

    ObjectInputStream in = new ObjectInputStream(new BufferedInputStream(new FileInputStream(file)));

    String[] uris = (String[]) in.readObject();

    classes = new Classe[uris.length];

    for (int i=0;i<uris.length;i++) {

      classes[i] = oimodel.findClasse(uris[i]);

      ClasseHash.put(classes[i],new Integer(i));

    }

    simMatrix = (double[][]) in.readObject();

  }



  /**

  * Saves a SimyMatrix to a file.

  */


  public void save(File file) {

    ObjectOutputStream out = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream(file)));

    String[] uris = new String[classes.length];

    for (int i=0;i<uris.length;i++) uris[i] = classes[i].findURI();

    out.writeObject(uris);

    out.writeObject(simMatrix);

    out.close();

  }


  public double findSimy(Classe clas1, Classe clas2) {

    int clas1Int = ((Integer)ClasseHash.find(clas1)).intValue();

    int clas2Int = ((Integer)ClasseHash.find(clas2)).intValue();

    return simMatrix[clas1Int][clas2Int];

  }



/** Returns the loaded ontology matcher class.
     *
     * @return Class, the loaded ontology matcher class.
     */


 public Class getAggregator() {

        Class mc = null;
```

```
            try    {

                if (aggName == null) {

                  Accumulate.print(System.out, "[AggFactory]", " Aggregator is not
given.", false);

                                                        aggName =
Accumulate.CBONTA;


                }

     mc = ONTOLOGYALSLosder.load("Ontology. Alignment. System..aggregator."
+ aggName + "Aggregator");

            } catch (Exception ex) {

                ex.printStackTrace();

            }

            return mc;

      }

public static AlignmentAggregator createAggregator(String type) throws
UnrecognisedTypeException {

/*this method creates an instance of the specified internal matcher that takes uri
for first ontology and uri for second ontology  as inputs */

            AlignmentAggregator ma = null;

            try    {

                Class c = (new AggregatorFactory(type)).getAggregator();

                ma = (AlignmentAggregator)c.getConstructor().newInstance();

            } catch (NoSuchMethodException nsme) {

                throw new UnrecognisedTypeException(type);

            } catch (IllegalAccessException iae) {

                throw new UnrecognisedTypeException(type);

            } catch (InstantiationException ie) {

                throw new UnrecognisedTypeException(type);

            } catch (InvocationtologyargetException ite) {

                ite.printStackTrace(Accumulate.logps);

                throw new UnrecognisedTypeException(type);

            }

            return ma;

      }

       public static AlignmentAggregator createAggregator() throws
UnrecognisedTypeException {

/*this method creates an instance of the specified internal matcher that takes uri
for first ontology and uri for second ontology  as inputs */

            AlignmentAggregator ma = null;

            try    {

                Class c = (new
AggregatorFactory(Accumulate.CBONTA)).getAggregator();

            } catch (InvocationtologyargetException ite) {

                ite.printStackTrace(Accumulate.logps);

                throw new UnrecognisedTypeException(Accumulate.CBONTA);

            }

            return ma;

       }

    public static void main( String[] args ){

    }

}

package Ontology.Alignment.System..aggregator;

import cern.colt.matrix.*;

import cern.colt.matrix.impl.*;

import java.util.Vec;

import java.util.Iterator;

public class AlignmentMatrixDouble extends AlignmentMatrix {

/* this method regarding AlignmentMatrixDouble for storing matching results.
Rows of the matrix are concepts from the target ontology while columns are
different matchers. While matrix corresponds to a concept from the source
ontology */

/*
 * AlignemtMatrix stores the similarity values of matching candidates.  There are
two types of
 * AlignemntMatrix: double matrix containing only double type elements and
Object matrix containing objects as elements.
 *
 */
 protected DoubleMatrix2D matrix;

 public AlignmentMatrixDouble(String id) {

                                super(id);

/** A two dimensions matrix for storing mapping results.
 * Rows of the matrix are concepts from the target ontology while columns are
different matchers, Each matrix corresponds to a concept from the source
ontology.
       *
       */

                }

       public AlignmentMatrixDouble(String id, int m, int n) {

                                super(id, m, n);

                }

public double getValue(String cp, String matcher) throws
IndexOutOfBoundsException {

                                int m, n;

                                if (column.contologyains(matcher)) {

                                                n = column.indexOf(matcher);

                                } else {

                                                n = column.size();

                                                column.add(matcher);

                                                if (n > cols) {

                                                                throw new
IndexOutOfBoundsException("Column size exceeds the size of the Matrix");

                                                }

                                }

                                if (row.contologyains(cp)) {

                                                m = row.indexOf(cp);
```

```
                } else {
                        m = row.size();
                        row.add(cp);
                        if (m > rows) {
                                throw new
IndexOutOfBoundsException("Row size exceeds the size of the Matrix");
                        }
                }
                return (double)matrix.get(m, n);
        }
        public double getValue(int m, int n) throws
java.lang.IndexOutOfBoundsException {
                double res = 0;
                if ((m <= rows) && (n <= cols)) {
                        res = (double)matrix.get(m, n);
                } else {        throw new
IndexOutOfBoundsException("Row or Column size exceeds the size of the
Matrix");
                }
                return res;
        }
        public void constructMatrix(int m, int n) {
                rows = m; cols = n;
                matrix = new DenseDoubleMatrix2D(m, n);
        }
public void setValue(String cp, String matcher, double value) throws
IndexOutOfBoundsException {
                int m, n;
                if (column.contologyains(matcher)) {
                        n = column.indexOf(matcher);
                } else {
                        n = column.size();
                        column.add(matcher);
                        if (n > cols) {
        throw new IndexOutOfBoundsException("Column size exceeds the
size of the Matrix");
                        }
                }
                if (row.contologyains(cp)) {
                        m = row.indexOf(cp);
                } else {
                        m = row.size();
                        row.add(cp);
                        if (m > rows) {                 throw new
IndexOutOfBoundsException("Row size exceeds the size of the Matrix");
                        }
                }
                                        }
public void setValue (int m, int n, double value) throws
java.lang.IndexOutOfBoundsException {
                if ((m <= rows) && (n <= cols)) {
                        matrix.set(m, n, value);
                } else {
        throw new IndexOutOfBoundsException("Row or Column size
exceeds the size of the Matrix");
                }
        public String toString() {
                StringBuffer sb = new StringBuffer();
                String[] rs = getRowNames();
                for (int i = 0; i < cs.length; i++) {
                        sb.append("\t"+cs[i]);
                }
                sb.append("\r\n");
                for (int i = 0; i < rs.length; i++) {
                        sb.append(rs[i]);
                        for (int j = 0; j < cs.length; j++) {
        sb.append("\t"+matrix.get(i, j));
                        }
                        sb.append("\r\n");
        }
                return sb.toString();
        }
        private String getWhiteSpace(int n) {
                StringBuffer sb = new StringBuffer();
                for (int i = 0; i < n; i++) {
                        sb.append(" ");
                }
                return sb.toString();
        }
        public static void main (String[] args) {
        }
}
package Ontology. Alignment. System.aggregator;
import cern.colt.matrix.*;
import cern.colt.matrix.impl.*;
import java.util.Vec;
import java.util.Iterator;
```

```
import Ontology. Alignment. System.util.*;

/* AlignmentAggregator interface defines a list of methods to be implemented by any concrete aggregators, among which getAggregationResult() returns the results of aggregation and initializingMatrix (String id, int m, int n) constructs an m × n matrix with name id. TypeIObjectAggregator initialise an aggregator containing MatchingUnit similarities from different matchers. */

public class AlignmentMatrixObject extends AlignmentMatrix {

/* this method regarding AlignmentMatrixObject for storing matching results. Rows of the matrix are concepts from the target ontology while columns are different matchers. While matrix corresponds to a concept from the source ontology */

/** A two dimensions Double type matrix for storing matching results.

*/
		class RelationScoPair {
				private String rel;
				private double sco;
				public RelationScoPair(String rel, double v) {
						this.rel = rel;
						sco = v;
				}
				public String getRelation () {
						return rel;
				}
				public double getSco () {
						return sco;
				}
				public String toString() {
						return rel+"|"+sco;
				}
		};
protected ObjectMatrix2D matrix;
		public AlignmentMatrixObject(String id) {
				super(id);
		}
	public AlignmentMatrixObject(String id, int m, int n) {
				super(id, m, n);
		}
 public MatchingUnit getValue(String cp, String matcher) throws IndexOutOfBoundsException {
				int m, n;
				if (column.contologyains(matcher)) {
						n = column.indexOf(matcher);
				} else {
						n = column.size();
						column.add(matcher);
						if (n > cols) {
throw new IndexOutOfBoundsException("Column size exceeds the size of the Matrix");
						}
				}
				if (row.contologyains(cp)) {
						m = row.indexOf(cp);
				} else {
						m = row.size();
						row.add(cp);
						if (m > rows) {
throw new IndexOutOfBoundsException("Row size exceeds the size of the Matrix");
						}
				}
return (MatchingUnit)matrix.get(m, n);
		}
 public MatchingUnit getValue(int m, int n) throws java.lang.IndexOutOfBoundsException {
				MatchingUnit res = null;
				if ((m <= rows) && (n <= cols)) {
						res = (MatchingUnit)matrix.get(m, n);
				} else {
throw new IndexOutOfBoundsException("Row or Column size exceeds the size of the Matrix");
				}
				return res;
		}
  public void constructMatrix(int m, int n) {
				rows = m; cols = n;
				matrix = new DenseObjectMatrix2D(m, n);
		}
  public boolean equals (AlignmentMatrix m) {
		return (name.equalsIgnoreCase(m.getMatrixName()))? true:false;
		}
  public void setValue(String cp, String matcher, MatchingUnit value) throws IndexOutOfBoundsException {
				int m, n;
				if (column.contologyains(matcher)) {
						n = column.indexOf(matcher);
				} else {
						n = column.size();
						column.add(matcher);
						if (n > cols) {
```

```
                throw new IndexOutOfBoundsException("Column size exceeds the
size of the Matrix");

                                }

                        }

                        if (row.contologyains(cp)) {

                                m = row.indexOf(cp);

                        } else {

                                m = row.size();

                                row.add(cp);

                                if (m > rows) {

                throw new IndexOutOfBoundsException("Row size exceeds the
size of the Matrix");

                                }

                        }

                                }

 public void setValue (int m, int n, MatchingUnit value) throws
java.lang.IndexOutOfBoundsException {

                        if ((m <= rows) && (n <= cols)) {

                                matrix.set(m, n, value);

                        } else {

throw new IndexOutOfBoundsException("Row or Column size exceeds the size of
the Matrix");

                        }

                }

public String toString() {

                        StringBuffer sb = new StringBuffer();

                        String[] rs = getRowNames();

                        String[] cs = getColumnNames();

                        sb.append("[Matrix] " + name + "\r\n\r\n");

                        sb.append(getWhiteSpace(cs[0].length()));

                        for (int i = 0; i < cs.length; i++) {

                                sb.append("\t"+cs[i]);

                        }

                        sb.append("\r\n");

                        for (int i = 0; i < rs.length; i++) {

                                sb.append(rs[i]);

                                for (int j = 0; j < cs.length; j++) {

                sb.append("\t"+matrix.get(i, j));

                                }

                                sb.append("\r\n");

                        }

                        return sb.toString();

                }

                private String getWhiteSpace(int n) {

                        StringBuffer sb = new StringBuffer();

                        for (int i = 0; i < n; i++) {

                                sb.append(" ");

                        }

                        return sb.toString();

                }

                        }

}
```

```
package Ontology. Alignment. System.aggregator;

import cern.colt.matrix.*;

import cern.colt.matrix.impl.*;

import java.util.Vec;

import java.util.Iterator;

   /* Alignment Matrix defines the data structure to hold matching results from
   different matchers. An instance of Alignment Matrix is illustrated in Figure
   5.10, with each row corresponding to a pair of entities from the source and
   target ontologies, and each column corresponding to a particular matcher.
   Each row of the resultant matrix is assigned a symbolic name for easy access.
   This method returns the names of rows (matching entity pairs) as an array
   of strings.  */

public abstract class AlignmentMatrix {

/*AlignmentMatrix stores the similarity values of matching candidates.*/

    protected Vec<String> row;

     protected int cols, rows;

    protected String name;

    public AlignmentMatrix(String id) {

                        name = id;

                        column = new Vec<String>();

                        row = new Vec<String>();

                        cols = 0;

                        rows = 0;

                }

public AlignmentMatrix(String id, int m, int n) {

/*  AlignmentMatrix with size (m, n) and name id*/

                        this(id);

                constructMatrix(m, n);

                }

    public String getMatrixName() {

                        return name;

                }

     public String getRowName (int m) {

                        return (row.size() > m)? row.elementAt(m):null;
```

```
                }

        public String getColumnName (int n) {

                                return (column.size() > n)?
column.elementAt(n):null;

                }

        public String[] getRowNames () {

                                String[] type = new String[row.size()];

                                return row.toArray(type);

                }

        public String[] getColumnNames () {

                                String[] type = new String[column.size()];

                                return column.toArray(type);

                }

        public void setColumnSize(int n) {

                                cols = n;

                }

        public void setRowSize(int m) {

                                rows = m;

                }

 abstract protected void constructMatrix(int m, int n);

public boolean equals (AlignmentMatrix m) {

                                return
(name.equalsIgnoreCase(m.getMatrixName()))? true:false;

}

}

package Ontology.Alignment. System.semanticFeature;

import conta.hp.hpl.jena.ontologyology.*;

import conta.hp.hpl.jena.rdf.model.*;

import conta.hp.hpl.jena.shared.PrefixMatching;

import conta.hp.hpl.jena.util.iterator.ExtendedIterator;

import java.util.*;

import java.io.*;

import java.text.*;

import java.net.URI;

import Ontology. Alignment.System. Exception.*;

import Ontology.Alignment. System.until.*;

   /*This system uses standard languages, such as OWL or RDF as input, which
   provide vocabularies to define the formal semantics of ontology. Thus, they
   use owl:Class and rdfs:subClassOf in order to define concepts and sub-
   concepts, and rdfs:Property and rdfs:subPropertyOf in order to define
   properties and sub-properties. They also use rdfs:domain and rdfs:range of a
   property to define what concepts can have the property and what instances
   of the concepts can be the values of the property. All these expressions help
   to extract element easily from ontologies.        */

/** This class extracts useful features to facilitate matching of ontological
entities.

 */

public class OWLUsefulMemFeature extends OWLUsefulFeature {

   private anonFilter afilter = new anonFilter();

    public OWLUsefulMemFeature(URI sourceURI) {

/*this method returns the full URI of the current ontologies. */

          super(sourceURI);

     }

     public OWLUsefulMemFeature(URI sourceURI, String[] ignored) {

          super(sourceURI, ignored);

     }

 public OWLUsefulMemFeature(String file, String ns) {

          super(file, ns);

     }

public OWLUsefulMemFeature(String file, String ns, String[] ignored) {

          super(file, ns, ignored);

    }

     public void output(String fileName) {

          try    {

               try                {

                    FileOutputStream fos = new FileOutputStream(fileName);

                    ontologyModel.write(fos);

                    fos.close();

               } catch (Exception ex) {

                    ex.printStackTrace();

                    throw new Exception("[OWLFMXtr] exceptions when outputing
" + ontologyId + " into local file " + fileName);

               }

          } catch (Exception ex) {

               ex.printStackTrace();

          }

    }/**

   * Allows to set the objet used to compute similarites. Please note that this

   * is necessary before you can call "calculate" on a SimyMatrix that

   * was loadet from disk.

   */

   public void setSimy(Simy sim) {

      this.sim = sim;

   }



   public void calculate(){

      for (int x=0;x<classes.length;x++) {

         Classe clas1 = classes[x];
```

```
        simMatrix[x][x] = 1.0;
        for (int y=x+1;y<classes.length;y++) {
          Classe clas2 = classes[y];
          simMatrix[x][y] = sim.calculateSimy(clas1,clas2);
        }
      }
      for (int x=0;x<classes.length;x++) {
        for (int y=0; y<x;y++) {
          simMatrix[x][y] = simMatrix[y][x];
        }
      }
    }
    public double[][] findSimMatrix() {
      return simMatrix;
    }
    public Classe[] findClasses() {
      return classes;
    }
public Vector findMostSimilar(Classe clas, double excludeThreshold, double
includeThreshold, int normalNumber, int maxNumber) {
      double[] sim = new double[maxNumber];
      Classe[] classes = new Classe[maxNumber];
      int present = ((Integer) ClasseHash.find(clas)).intValue();
      for (int i=0;i<classes.length;i++) {
        if (present != i) {
          double sim = simMatrix[present][i];
          addComparison(classes[present],sim,sim,classes);
        }
      }

      Vector toReturn = new Vector(maxNumber);
      for (int i=0;i<sim.length;i++) {
        if ((i<normalNumber) && (sim[i] > excludeThreshold)) {
          toReturn.addElement(classes[i]);
        }
        else if ((i<maxNumber) && (sim[i] > includeThreshold)) {
          toReturn.addElement(classes[i]);
        }
        else {
          break;
        }
      }
      return toReturn;
    }

      public Iterator extractAllClasses() {
/*this method returns an iterator in excess of all non-anonymous classes
excluding the top */
      AbstractList<OntologyClass> cls = new Vec<OntologyClass> ();
      Iterator j = ontologyModel.listClasses();
      for (ExtendedIterator i = ontologyModel.listClasses().filterDrop(afilter);
i.hasNext(); ) {
        OntologyClass c = (OntologyClass) i.next();
        if (c.equals(ontologyModel.getProfile().THING())) {
          contologyinue;
        }
        cls.add(c);
      }
      return cls.iterator();
    }
  public Iterator extractAllClassNames() {
/*this method returns an iterator in excess of all non-anonymous  name classes
excluding the top */
          AbstractList<String> clsNames = new Vec<String>();
          for (Iterator i = extractAllClasses(); i.hasNext(); ) {
              OntologyClass c = (OntologyClass) i.next();
              String name = c.getURI();
              if (!clsNames.contologyains(name)) {
                  clsNames.add(name);
              }
          }
          return clsNames.iterator();
    }
    public Iterator extractAllClassStr() {
          AbstractList<Object> cstrs = new Vec<Object>();
          for (Iterator i = extractAllClasses(); i.hasNext(); ) {
              OntologyClass c = (OntologyClass) i.next();
              String name = c.getURI();
              String label = c.getLabel(null);
              if (label == null) {
                  if (name.indexOf("%26") < 0) {
```

```
                    label = c.getLocalName();
                } else {
                    name.replaceAll("%26", "and");
                    label = Accumulate.cleanUpName(name);
                }
            }
            if ( (name != null)&&(label != null) ) {
                OntologyResourceStr ors = new OntologyResourceStr(name,
label);
                if (!cstrs.contologyains(ors)) {
                    cstrs.add(ors);
                }
            }
        }
        return cstrs.iterator();
    }
    public Iterator extractSuperClasses (String clsName, boolean direct) throws
OntologyResourceNotFoundException {
/*this mthod returns an iterator over all super classes of a given class */
        OntologyClass cls = ontologyModel.getOntologyClass(clsName);
        if (cls!=null) {
            return extractSuperClasses(cls, direct);
        } else {
            throw new OntologyResourceNotFoundException(clsName,
ontologyId);
        }
    }
    public Iterator extractSuperClasses (OntologyClass cls, boolean direct) {
        AbstractList<OntologyClass> supers = new Vec<OntologyClass>();
        for (ExtendedIterator i = cls.listSuperClasses(direct).filterDrop(afilter);
i.hasNext(); ) {
            OntologyClass c = (OntologyClass) i.next();
            supers.add(c);
        }
        return supers.iterator();
    }
    public Iterator extractSuperClassStr (String clsName, boolean direct) throws
OntologyResourceNotFoundException {
        AbstractList<OntologyResourceStr> supers = new
Vec<OntologyResourceStr>();
        for (Iterator i = extractSuperClasses(clsName, direct); i.hasNext(); ) {
            OntologyClass c = (OntologyClass)i.next();
            String label = c.getLabel(null);
            String name = c.getURI();
            if (label == null) {
                if (name.indexOf("%26") < 0) {
                    label = c.getLocalName();
                } else {
                    name.replaceAll("%26", "and");
                    label = Accumulate.cleanUpName(name);
                }
            }
            if (label != null) {
                OntologyResourceStr str = new OntologyResourceStr(name,
label);
                if (supers.contologyains(str)) {
                    contologyinue;
                }
                supers.add(str);
            }
        }
        return supers.iterator();
}
    public Iterator extractSubClasses (String clsName, boolean direct)
throws OntologyResourceNotFoundException {
/*this method returns an iterator over all sub classes of a given class */
        OntologyClass cls = ontologyModel.getOntologyClass(clsName);
        if (cls!=null) {
            return extractSubClasses(cls, direct);
        } else {
            throw new OntologyResourceNotFoundException(clsName);
        }
    }
    public Iterator extractSubClasses (OntologyClass cls, boolean direct) {
        AbstractList<OntologyClass> subs = new Vec<OntologyClass>();
        if (!isInCurrentModel(cls)) {
            return (new Vec()).iterator();
        }
        for (ExtendedIterator i = cls.listSubClasses(direct).filterDrop(afilter);
i.hasNext(); ) {
            OntologyClass s = (OntologyClass) i.next();
            subs.add(s);
        }
        return subs.iterator();
    }
/** Returns an iterator over all sub classes of a given class
     */
```

```
    public Iterator extractSubClassStr (String clsName, boolean direct) throws
OntologyResourceNotFoundException {

        AbstractList<OntologyResourceStr> subs = new
Vec<OntologyResourceStr>();

        for (Iterator i = extractSubClasses(clsName, direct); i.hasNext(); ) {

            OntologyClass c = (OntologyClass)i.next();

            String label = c.getLabel(null);

            String name = c.getURI();

            if (label == null) {

                if (name.indexOf("%26") < 0) {

                    label = c.getLocalName();

                } else {

                    name.replaceAll("%26", "and");

                    label = Accumulate.cleanUpName(name);

                }

            }

            if (label != null) {

                OntologyResourceStr str = new OntologyResourceStr(name,
label);

                if (subs.contologyains(str)) {

                    contologyinue;

                }

                subs.add(str);

            }

        }

        return subs.iterator();

    }

    public Iterator extractSameAsNames (OntologyResource r) {

/* returen value  true if the two resources are declared as owl:sameAs; return rhe
classes which are declared to be the equivlant otherwise, answers false */

        AbstractList<String> sameAs = new Vec<String>();

        for (Iterator i = extractSameAs(r); i.hasNext(); ) {

            String resourceName = ((OntologyResource)i.next()).getURI();

            if (!sameAs.contologyains(resourceName)) {

                sameAs.add(resourceName);

            }

        }

        return sameAs.iterator();    }

 public Iterator extractSameAsNames (String r) throws
UnrecognizedNameException {

        OntologyResource res = ontologyModel.getOntologyResource(r);

        if (res != null) {

            return extractSameAsNames(res);

        } else {

            throw new UnrecognizedNameException (this, r);

        }

    }

 public boolean isSameAs (String s, String t) {

    boolean res = false;

    try    {

        for (Iterator i = extractSameAsNames(s); i.hasNext(); ) {

            if (t.equals((String)i.next())) {

                res = true;

                break;

            }

        }

        for (Iterator i = extractSameAsNames(t); i.hasNext(); ) {

            if (s.equals((String)i.next())) {

                res = true;

                break;

            }

        }

    } catch (UnrecognizedNameException une) {

        res = false;

    }

    return res;

 }

  public Iterator extractInstances (OntologyClass cls) {

    AbstractList<Indiv> instances = new Vec<Indiv>();

    for (Iterator i = cls.listInstances(); i.hasNext(); ) {

        Indiv ind = (Indiv)i.next();

        if (!instances.contologyains(ind)) {

            instances.add(ind);

        }

    }

    return instances.iterator();

 }

   public Iterator extractInstances (String clsName) throws
OntologyAlignment.System.Exception.OntologyResourceNotFoundException {

    OntologyClass cls = ontologyModel.getOntologyClass(clsName);

    if (cls != null) {

        return extractInstances(cls);

    } else {

        throw new OntologyResourceNotFoundException (clsName);

    }
```

```
    }

    public Iterator extractProp (OntologyClass cls, boolean direct) {

        AbstractList<OntologyProperty> Prop = new Vec<OntologyProperty>();

        for (Iterator i = cls.listSuperClasses(direct); i.hasNext(); ) {

            OntologyClass c = (OntologyClass)i.next();

            if (c.isRestriction()) {

                Prop.add(c.asRestriction().getOnProperty());

            }

        }

        return Prop.iterator();

    }

    public Iterator extractProp (String clsName, boolean direct) throws
OntologyResourceNotFoundException {

        OntologyClass cls = ontologyModel.getOntologyClass(clsName);

        if (cls != null) {

            return extractProp(cls, direct);

        } else {

            throw new OntologyResourceNotFoundException(clsName);

        }

    }

    public Iterator extractPropertyStr (String clsName, boolean direct) throws
OntologyResourceNotFoundException {

        AbstractList<OntologyResourceStr> props = new
Vec<OntologyResourceStr>();

/* this class returns an iterator over properties that are declared properties of a
given class */

        for (Iterator i = extractProp(clsName, direct); i.hasNext(); ) {

            OntologyProperty r = (OntologyProperty)i.next();

            String label = r.getLabel(null);

            String name = r.getURI();

            if (label == null) {

                if (name.indexOf("%26") < 0) {

                    label = r.getLocalName();

                } else {

                    name.replaceAll("%26", "and");

                    label = Accumulate.cleanUpName(name);

                }

            }

            if (label != null) {

                OntologyResourceStr str = new OntologyResourceStr(name,
label);

                if (!props.contologyains(str)) {

                    props.add(str);

                }

            }

        }

        return props.iterator();

    }

    protected String extractPropertyName (OntologyProperty pro) {

        return pro.getURI();

    }

    public Iterator extractAllProp () {

        AbstractList<OntologyProperty> pros = new Vec<OntologyProperty>();

        for (Iterator i = ontologyModel.listObjectProp(); i.hasNext(); ) {

            pros.add((OntologyProperty)i.next());

        }

        for (Iterator i = ontologyModel.listDatatypeProp(); i.hasNext(); ) {

            pros.add((OntologyProperty)i.next());

        }

        return pros.iterator();

    }

    public Iterator extractAllPropertyStr () {

        AbstractList<Object> res = new Vec<Object>();

        AbstractList<OntologyProperty> props = new Vec<OntologyProperty>();

        for (Iterator i = ontologyModel.listObjectProp(); i.hasNext(); ) {

            props.add( ((OntologyProperty)i.next()) );

        }

        for (Iterator i = ontologyModel.listDatatypeProp(); i.hasNext(); ) {

            props.add( ((OntologyProperty)i.next()) );

        }

        for (Iterator i = props.iterator(); i.hasNext(); ) {

            OntologyProperty pro = (OntologyProperty)i.next();

            String label = pro.getLabel(null);

            String name = pro.getURI();

            if (label == null) {

                if (name.indexOf("%26") < 0) {

                    label = pro.getLocalName();

                } else {

                    name.replaceAll("%26", "and");

                    label = Accumulate.cleanUpName(name);

                }

            }

            if (label != null) {

                OntologyResourceStr str = new OntologyResourceStr(name,
label);
```

```
                    if (!res.contologyains(str)) {
                        res.add(str);
                    }
                }
            }
            return res.iterator();
        }
    public Iterator extractSuperProperty (OntologyProperty pro) {
            AbstractList<OntologyProperty> pros = new Vec<OntologyProperty>();
            for (Iterator i = pro.listSuperProp(); i.hasNext(); ) {
                OntologyProperty p = (OntologyProperty)i.next();
                if (!pros.contologyains(p)) {
                    pros.add(p);
                }
            }
            return pros.iterator();
        }
      public Iterator extractSuperProperty (String proName) throws
OntologyResourceNotFoundException {
/* this class returns an iterator over properties that are super properties of the
current property */
            OntologyProperty pro = ontologyModel.getOntologyProperty(proName);
            if (pro != null) {
                return extractSuperProperty(pro);
            } else {
                throw new OntologyResourceNotFoundException(proName);
            }
        }
      public Iterator extractSuperPropertyStr (String proName) throws
OntologyResourceNotFoundException {
            AbstractList<OntologyResourceStr> pros = new
Vec<OntologyResourceStr>();
            OntologyProperty pro = ontologyModel.getOntologyProperty(proName);
            if (pro != null) {
                for (Iterator i = pro.listSuperProp(); i.hasNext(); ) {
                    OntologyProperty p = (OntologyProperty)i.next();
                    String label = p.getLabel(null);
                    String name = p.getURI();
                    if (label == null) {
                        if (name.indexOf("%26") < 0) {
                            label = p.getLocalName();
                        } else {
                            name.replaceAll("%26", "and");
                            label = Accumulate.cleanUpName(name);
                        }
                    }
                    OntologyResourceStr str = new OntologyResourceStr(name,
label);
                    if (!pros.contologyains(str)) {
                        pros.add(str);
                    }
                }
            } else {
                throw new OntologyResourceNotFoundException(proName);
            }
            return pros.iterator();
        }
      public Iterator extractSubProperty (OntologyProperty pro) {
            AbstractList<OntologyProperty> pros = new Vec<OntologyProperty>();
            for (Iterator i = pro.listSubProp(); i.hasNext(); ) {
                OntologyProperty p = (OntologyProperty)i.next();
                if (!pros.contologyains(p)) {
                    pros.add(p);
                }
            }
            return pros.iterator();
        }
      public Iterator extractSubProperty (String proName) throws
OntologyResourceNotFoundException {
/* this class returns an iterator over properties that are sub properties of the
current property */
            OntologyProperty pro = ontologyModel.getOntologyProperty(proName);
            if (pro != null) {
                return extractSubProperty(pro);
            } else {
                throw new OntologyResourceNotFoundException(proName);
            }
        }
      public Iterator extractSubPropertyStr (String proName) throws
OntologyResourceNotFoundException {
            AbstractList<OntologyResourceStr> pros = new
Vec<OntologyResourceStr>();
            OntologyProperty pro = ontologyModel.getOntologyProperty(proName);
            if (pro != null) {
                for (Iterator i = pro.listSubProp(); i.hasNext(); ) {
                    OntologyProperty p = (OntologyProperty)i.next();
                    String label = p.getLabel(null);
```

```
                String name = p.getURI();
                if (label == null) {
                    if (name.indexOf("%26") < 0) {
                        label = p.getLocalName();
                    } else {
                        name.replaceAll("%26", "and");
                        label = Accumulate.cleanUpName(name);
                    }
                }
                OntologyResourceStr str = new OntologyResourceStr(name,
label);
                if (!pros.contologyains(str)) {
                    pros.add(str);
                }
            }
        } else {
            throw new OntologyResourceNotFoundException(proName);
        }
        return pros.iterator();       }
    public Iterator extractInverseProperty (OntologyProperty pro, boolean
extended) {
        AbstractList<OntologyProperty> pros = new Vec<OntologyProperty>();
        for (Iterator i = pro.listInverse(); i.hasNext(); ) {
            OntologyProperty p = (OntologyProperty)i.next();
            if (!pros.contologyains(p)) {
                pros.add(p);
            }
        }
        if (extended) {
            for (Iterator i = pro.listInverseOf(); i.hasNext(); ) {
                OntologyProperty p = (OntologyProperty)i.next();
                if (!pros.contologyains(p)) {
                    pros.add(p);
                }
            }
        }
        return pros.iterator();
    }
    public Iterator extractInverseProperty (String proName, boolean extended)
throws OntologyResourceNotFoundException {
/* this classReturns an iterator over properties that are declared inverse
properties of a given class */
        OntologyProperty pro = ontologyModel.getOntologyProperty(proName);
        if (pro != null) {
            return extractInverseProperty(pro, extended);
        } else {
            throw new OntologyResourceNotFoundException (proName);
        }
    }
    public Iterator extractPropertyRange(OntologyProperty pro, boolean
extended) {
        AbstractList<Resource> range = new Vec<Resource>();
        try    {
            for (ExtendedIterator i = pro.listRange().filterDrop(afilter);
i.hasNext(); ) {
                Resource r = (Resource)i.next();
                if (range.contologyains(r)) {
                    contologyinue;
                }
                range.add(r);
            }
        } catch (ProfileException ope) {
            ope.printStackTrace();
        }
        try    {
            if (extended) {
                for (Iterator i = pro.listDeclaringClasses(false); i.hasNext(); ) {
                    OntologyClass c = (OntologyClass)i.next();
                    for (Iterator j = c.listSuperClasses(true); j.hasNext(); ) {
                        OntologyClass sc = (OntologyClass)j.next();
                        if (sc.isRestriction()) {
                            Restriction r = sc.asRestriction();
                            if (r.onProperty(pro)) {
                                if (r.isAllValuesFrontaRestriction()) {
                                    Resource avf =
r.asAllValuesFrontaRestriction().getAllValuesFronta();
                                     if (avf.isAnon()) contologyinue;
                                    if (!range.contologyains(avf)) {
                                        range.add(avf);
                                    }
                                }
                                if (r.isSontaeValuesFrontaRestriction()) {
                                    Resource svf =
r.asSontaeValuesFrontaRestriction().getSontaeValuesFronta();
                                    if (svf.isAnon()) contologyinue;
                                    if (!range.contologyains(svf)) {
```

```
                                        range.add(svf);
                                    }
                                }
                            }
                        }
                    }
                }
            } catch (ProfileException ope) {
                ope.printStackTrace();
            }
            return range.iterator();
    }
    public Iterator extractPropertyDomain (String proName, boolean extended)
throws OntologyResourceNotFoundException {
/* this method  Returns an iterator over the domain of a specified property */
            AbstractList<Resource> Domain = new Vec<Resource>();
            OntologyProperty pro = ontologyModel.getOntologyProperty(proName);
            if (pro != null) {
                for (Iterator i = extractPropertyDomain(pro, extended); i.hasNext();
) {
                    Domain.add( (Resource)i.next() );
                }
            } else {
                throw new OntologyResourceNotFoundException (proName,
ontologyId);
            }
            return Domain.iterator();
    }
    public Iterator extractPropertyDomain (OntologyProperty pro, boolean
extended) {
            AbstractList<Resource> Domain = new Vec<Resource>();
              try {
                for (ExtendedIterator i = pro.listDomain().filterDrop(afilter);
i.hasNext(); ) {
                    Resource r = (Resource)i.next();
                    if (Domain.contologyains(r)) {
                        contologyinue;
                    }
                    Domain.add(r);
                }
            } catch (ProfileException ope) {
                ope.printStackTrace();
            }
                try             {
                if (extended) {
                    for (ExtendedIterator i =
pro.listDeclaringClasses().filterDrop(afilter); i.hasNext(); ) {
                        OntologyClass r = (OntologyClass)i.next();
                        if (Domain.contologyains(r)) {
                            contologyinue;
                        }
                        for (Iterator j = r.listSuperClasses(true); j.hasNext(); ) {
                            OntologyClass rj = (OntologyClass)j.next();
                            if (rj.isRestriction()) {
                                if (rj.asRestriction().onProperty(pro)) {
                                    if (Domain.contologyains(r)) {
                                        contologyinue;
                                    }
                                    Domain.add(r);
                                }
                            private static void addComparison(Classe present,
double sim, double[] sim, Classe[] classes) {
    if (sim > sim[sim.length-1]) {
      for (int i=0;i<sim.length;i++) {
        if (sim > sim[i]) {
          moveBack(sim,classes,i);
          sim[i] = sim;
          classes[i] = present;
          break;
        }
      }
    }
  }
  private static void moveBack(double[] sim, Classe[] classes, int index) {
    for (int i=sim.length-1;i>index;i--) {
      sim[i] = sim[i-1];
      classes[i] = classes[i-1];
    }
  }  }
                        }
                    }
                }
            } catch (ProfileException pe) {
                pe.printStackTrace();
```

```
            }

            return Domain.iterator();

        }

        public Iterator extractAllSuperClasses(String clsName, boolean extended)
throws Exception {

/*this method returns an iterator over all super classes of a given class. */

            if (!extended) {

                return extractSuperClasses(clsName, false);

            } else {

                OntologyClass c = ontologyModel.getOntologyClass(clsName);

                AbstractList<OntologyClass> sup = new Vec<OntologyClass>();

                for (Iterator i = extractSuperClasses(c, false); i.hasNext(); ) {

                    sup.add((OntologyClass)i.next());

                }

                for (Iterator i = extractEquivClasses(c, true); i.hasNext(); ) {

                    OntologyClass ec = (OntologyClass)i.next();

                    for (Iterator j = extractSuperClasses(ec, false); j.hasNext(); ) {

                        OntologyClass sec = (OntologyClass)j.next();

                        if (sec.isAnon()) {

                            contologyinue;

                        }

                        if (sup.contologyains(sec)) {

                            contologyinue;

                        }

                        sup.add(sec);

                    }

                }

                return sup.iterator();

            }

        }

            public Iterator extractAllSubClasses(String clsName, boolean
extended) throws Exception {
            if (!extended) {

                return extractSubClasses(clsName, false);

            } else {

                OntologyClass c = ontologyModel.getOntologyClass(clsName);

                AbstractList<OntologyClass> sub = new Vec<OntologyClass>();

                for (Iterator i = extractSubClasses(c, false); i.hasNext(); ) {

                    sub.add((OntologyClass)i.next());

                }

                for (Iterator i = extractEquivClasses(c, true); i.hasNext(); ) {

                    OntologyClass ec = (OntologyClass)i.next();

                    if (ec.isAnon()) {

                        contologyinue;

                    }

                    for (Iterator j = extractSubClasses(ec, false); j.hasNext(); ) {

                        OntologyClass sec = (OntologyClass)j.next();

                        if (sec.isAnon()) {

                            contologyinue;

                        }

                        if (sub.contologyains(sec)) {

                            contologyinue;

                        }

                        sub.add(sec);

                    }

                }

                return sub.iterator();

            }   }

        public Iterator extractDomainAndRangeOfProperty(String proName) {

/* this method answers an iterator of strings in the format of PropName(Domain,
Range) as required by Structure-based matchers */

            AbstractList<String> triples = new Vec<String>();

            try    {

                for (Iterator i = extractPropertyDomain(proName, true); i.hasNext();
) {

                    String d = ((Resource)i.next()).getURI();

                    for (Iterator j = extractPropertyRange(proName, true);
j.hasNext(); ) {

                        String triple = proName + ", " + d + ", " +
((Resource)j.next()).getURI();

                        if (!triples.contologyains(triple)) {

                            triples.add(triple);

                        }

                    }

                }

            } catch (Exception ex) {

                ex.printStackTrace();

            }

            return triples.iterator();

        }

        public Iterator extractPropertyDomainStr(String proName) {

            Vec<OntologyResourceStr> Domains = new Vec<OntologyResourceStr>
();

            try    {

                for (Iterator i = extractPropertyDomain(proName, true); i.hasNext();
) {

                    OntologyResource r = (OntologyResource)i.next();
```

```
                String label = r.getLabel(null);
                String name = r.getURI();
                if (label == null) {
                    if (name.indexOf("%26") < 0) {
                        label = r.getLocalName();
                    } else {
                        name.replaceAll("%26", "and");
                        label = Accumulate.cleanUpName(name);
                    }
                }
                if (label != null) {
                    OntologyResourceStr str = new OntologyResourceStr(name,
label);
                    if (Domains.contologyains(str)) {
                        contologyinue;
                    }
                    Domains.add(str);
                }
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return Domains.iterator();
    }
    public Iterator extractPropertyRangeStr(String proName) {
        Vec<OntologyResourceStr> ranges = new Vec<OntologyResourceStr> ();
        try    {
            for (Iterator i = extractPropertyRange(proName, true); i.hasNext(); )
{
                OntologyResource r = (OntologyResource)i.next();
                String label = r.getLabel(null);
                String name = r.getURI();
                if (label == null) {
                    if (name.indexOf("%26") < 0) {
                        label = r.getLocalName();
                    } else {
                        name.replaceAll("%26", "and");
                        label = Accumulate.cleanUpName(name);
                    }
                }
                if (label != null) {
                    OntologyResourceStr str = new OntologyResourceStr(name,
label);
                    if (!ranges.contologyains(str)) {
                        ranges.add(str);
                    }
                }
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return ranges.iterator();
    }
    public static void main( String[] args ){
        try    {
            OWLUsefulMemFeature xtractor = new
OWLUsefulMemFeature(new URI(args[0]));
            System.out.println("results ");
            for (Iterator i = xtractor.extractAllClassStr(); i.hasNext(); ) {
                OntologyResourceStr r = (OntologyResourceStr) i.next();
                System.out.println("\n[class name] " + r);
                if (xtractor.hasSuperClass(r.getID())) {
                    System.out.println("\n[hasSuper] " + r + "has super
classes");
                }
                        }
        } catch (Exception e) {
            e.printStackTrace();
    }    } }
```